UNIVERSIDADE FEDERAL FLUMINENSE

DAVID BATISTA CARVALHO

Planejamento com Simulação de Percepções e Geração de Ruídos Dinâmicos para Storytelling Emergente

NITERÓI

2012

UNIVERSIDADE FEDERAL FLUMINENSE

DAVID BATISTA CARVALHO

Planejamento com Simulação de Percepções e Geração de Ruídos Dinâmicos para Storytelling Emergente

Dissertação de Mestrado apresentada ao Programa de Pós-Graduação em Computação da Universidade Federal Fluminense como requisito parcial para a obtenção do Grau de Mestre em Computação. Área de concentração: Computação Visual

Orientador:
Esteban Walter Gonzalez Clua

NITERÓI

2012

DAVID BATISTA CARVALHO

Planejamento com Simulação de Percepções e Geração de Ruídos Dinâmicos para Storytelling Emergente

Dissertação de Mestrado apresentada ao Programa de Pós-Graduação em Computação da Universidade Federal Fluminense como requisito parcial para a obtenção do Grau de Mestre em Computação. Área de concentração: Computação Visual

BANCA EXAMINADORA

Prof. Dsc. Esteban Walter Gonzalez Clua / IC-UFF
(Orientador)

Prof. Dsc. Marcos de Oliveira Lage Ferreira / IC-UFF

Prof. Dsc. Angelo Ernani Maia Ciarlini / UNIRIO

Niterói
2012

"Caia sete vezes, levante-se oito".
(Provérbio japonês)

Para minha família, amigos e mestres, que confiaram em mim.

Resumo

Digital storytelling vem se desenvolvendo como uma área que propõe técnicas e abordagens para a criação e apresentação de histórias por meios computacionais. Nesse sentido, diversas pesquisas da área têm visado o desenvolvimento de sistemas capazes de gerar narrativas que possam ser consideradas interessantes do ponto de vista humano. Entretanto, por ser difícil formalizar o conceito de história interessante, essas pesquisas têm concentrado esforços na aplicação de elementos característicos da Literatura ou na simulação de personagens e mundos virtuais em processos de geração de narrativas. Nessa mesma linha, este trabalho se propõe a viabilizar a execução proposital de ações que possam ser consideradas erradas, mas que venham a enriquecer as atuações dos personagens, através de um modelo de simulação de percepções. Para este fim, foi desenvolvido um processo de geração de histórias que, partindo de uma descrição que envolva as características físicas dos elementos que compõem o mundo onde a história se passa, gera e insere dinamicamente informações incorretas nos conhecimentos que os personagens possuem do mundo de forma coerente com seus pontos de vista. Estes conhecimentos são mapeados pelo processo em domínios e problemas de planejamento em PDDL, que são utilizados com o planejador SGPlan para a seleção de sequências de ações e eventos que representem histórias.

Palavras-chave: Storytelling, Percepção, Erro, Personagens, Geração de histórias, Ponto de vista.

Abstract

Digital storytelling has been developed as an area that proposes techniques and approaches to the creation and presentation of stories by computational means. In this sense, several research in the area have aimed to the development of systems capable of generating narratives that may be considered interesting in the human point of view. However, because it is difficult to formalize the concept of interesting story, these research studies have concentrated efforts in the application of elements characteristic of the Literature or on the simulation of characters and virtual worlds in narrative generation processes. In this same line, this work proposes to allow the purposeful execution of actions that may be considered wrong, but that may enrich the performance of the characters, through the use of a perception simulation model. To this end, it was developed a story generation process that, starting from a description that involves the physical characteristics of the elements that compose the world where the story takes place, dynamically generates and inserts incorrect information in the knowledge that the characters have about the world coherently with their point of view. This knowledge is mapped by the process into planning domains and problems in PDDL that are utilized with the planner SGPlan to the selection of action sequences that represent stories.

Keywords: Storytelling, Perception, Error, Characters, Story Generation, Point of view.

Lista de Figuras

2.1 Domínio das docas 6

2.2 Exemplo de especificação do domínio das docas. 7

2.3 Exemplo de especificação do problema. 7

2.4 Especificação de parte do domínio das docas em representação clássica, conforme [11]. 10

2.5 Especificação do problema. 11

2.6 Exemplo de HTN 15

2.7 Interface do Logtell para a geração de histórias 18

2.8 Exemplo de dramatização 18

2.9 Exemplo de execução do Façade 19

2.10 Curva de tensão aristotélica 20

3.1 Linha do tempo em turnos 25

3.2 Diagrama de classes usado para a inserção de ruídos. 28

3.3 Representação de um estado do mundo como um problema PDDL. 29

3.4 Exemplo de mapa 29

3.5 Diagrama de processos de um turno com inserção de ruídos 31

3.6 Domínio usado para a descrição do mundo. 34

4.1 Diagrama de classes usado para a simulação de percepção 41

4.2 Domínio criado automaticamente englobando os atributos do personagem. 43

4.3 Problema criado automaticamente englobando os atributos do personagem. 43

4.4 Problema criado automaticamente com a especificação do mapa. 45

4.5 Descrição parcial de um personagem 46

4.6 Diagrama dos processos de observação e interpretação, respectivamente . . 53
4.7 Exemplo de percepção com duplicação de ID 58
4.8 Mapa criado para a história da Chapeuzinho Vermelho 59
4.9 História gerada automaticamente pelo sistema 62
A.1 Descrição do tipo de objeto cesta 75
A.2 Descrição do modelo cesta de doces 75
A.3 Descrição do objeto cesta de doces 76
A.4 Descrição do mapa 77
A.5 Arquivo auxiliar de descrição de personagens 78
A.6 Descrição dos modelos de personagens 79
A.7 Descrição do personagem Caçador 81
A.8 Descrição da personagem Avó 83
A.9 Descrição do personagem Lobo 85
A.10 Descrição da personagem Chapeuzinho 87
A.11 Descrição dos eventos 88

Lista de Tabelas

4.1 Componentes dos personagens 60

4.2 Atributos dos personagens 60

Lista de Abreviaturas e Siglas

BDI	:	Beliefs-Desires-Intentions;
HTN	:	Hierarchical Task Network;
IA	:	Inteligência Artificial;
IPC	:	International Planning Competition;
PDDL	:	Planning Domain Definition Language;
SGPlan	:	Subgoal Partitioning and Global Search Planner;

Sumário

1 Introdução **1**

1.1 Motivação 2

1.2 Proposição 3

1.3 Contribuições 3

1.4 Estrutura 4

2 Fundamentação Teórica **5**

2.1 Planejamento e Representação de Problemas 5

2.2 Geração e Representação de Mundo 11

2.2.1 Character-based Storytelling 14

2.2.2 Plot-based Storytelling 16

2.2.3 Abordagem Híbrida 19

2.2.4 Orientando o Processo de Planejamento 21

3 Planejamento com Ruído Dinâmico para Storytelling Emergente **23**

3.1 Arquitetura 24

3.1.1 Linha do Tempo 24

3.1.2 Representação do Mundo 27

3.2 Ruído 30

3.2.1 Ruído por Omissão 31

3.2.2 Ruído por Divergência 32

3.3 Testes 33

3.3.1 Cenário 33
3.3.2 Resultados 35
3.4 Discussão 36

4 Simulação de Percepções **38**
4.1 Arquitetura 38
4.1.1 Representação do mundo 39
4.1.1.1 A criação do mundo virtual 41
4.1.2 Eventos e Comportamentos 46
4.2 Trabalhando com Erros de Percepção 49
4.2.1 Percepção 50
4.2.2 Tratamento de Erros 55
4.3 Testes 59
4.3.1 Cenário 59
4.3.2 Resultados 61
4.4 Discussão 64

5 Conclusão **67**
5.1 Considerações finais 67
5.2 Análise das Contribuições 68
5.3 Trabalhos futuros 69

Referências **72**

Apêndice A – Descrição do Mundo Virtual **75**

Capítulo 1

Introdução

Histórias são parte da vivência humana, criadas com experiências vividas ou elementos imaginários, no intuito de ensinar, entreter ou mesmo como forma de guardar memórias. Embora a tarefa de criar uma história por vezes seja considerada inerente ao ser humano, a possibilidade de geração automática de histórias tem sido buscada pela sua evidente utilidade na área de entretenimento digital: histórias usadas em filmes, livros, jogos eletrônicos como simples plano de fundo motivacional passariam a ter um maior nível de interação, podendo ser adaptadas às decisões e ações de seus usuários.

O digital storytelling surgiu como um novo paradigma para tratamento de histórias com o uso do computador. As pesquisas da área têm se concentrado em: representação, geração e apresentação. A representação trata do desenvolvimento de linguagens e modelos que representem as estruturas temporais e os elementos que compõem uma história, tais como os personagens e suas relações [19, 30]. A principal preocupação da geração é, dado um ambiente, um grupo de personagens, ações e eventos, determinar uma sequência de ações e eventos coerentes que possam ser interpretados como uma história interessante [13, 5]. Por fim, a apresentação trata da exibição da história de forma a instigar sentimentos [1], seja em um ambiente 3D através de técnicas de cinematografia [15], ou em um ambiente 2D através de um narrador [32]. Embora seja considerado bastante promissor, o storytelling ainda se encontra incipiente. "Nós podemos estar nos primeiros estágios do desenvolvimento de uma nova série de experiências interativas - um novo gênero de jogos ou uma nova mídia narrativa - ou, partindo de uma visão de certa forma pessimista, lutando contra desafios conceituais e tecnológicos que possam vir a se mostrar insolúveis", como dito por Swartjes em [29].

1.1 Motivação

Dentro da área de geração, o processo de criação de histórias tem sido desenvolvido através de algoritmos de planejamento provenientes da área de IA (Inteligência Artificial). O problema desta abordagem, no contexto de storytelling, está no determinismo característico desses algoritmos [22, 21], que, quando usados para simular a atuação dos personagens, geram histórias curtas onde eles alcançam seus objetivos da forma mais direta e correta possível, sem demonstrar personalidade ou fazer escolhas erradas, o que no mundo real é característico e inerente ao comportamento humano.

A tarefa de desenvolver histórias interessantes tem feito pesquisas dessa área se concentrarem em conferir credibilidade aos personagens, influenciando os algoritmos de planejamento através do uso de características humanas tais como: personalidade [28], emoções [31] e valores morais [8], ou ainda de elementos literários, como arcos de tensão [4, 16], que possam guiar estes algoritmos a escolher quais ações são mais propensas a serem executadas por quais personagens em que momentos da história.

Por meio destas características é possível criar histórias nas quais, por exemplo, um herói evita matar um vilão em virtude de seus valores morais, por mais que este seja o modo mais simples de resgatar uma princesa. Entretanto, elas não viabilizam um comportamento comum e que se mostra decisivo em histórias clássicas, tais como Romeu e Julieta, Chapeuzinho Vermelho e Cinderela: os personagens erram. Seja por não terem onisciência, por terem sido enganados por um vilão, não terem raciocínio perfeito ou simplesmente por não serem capazes de discernir um lobo de uma senhora, personagens costumam determinar o desenrolar de muitas histórias por tomarem ações precitadas. O erro é um elemento recorrente e por vezes a motivação da história, conferindo credibilidade e servindo como instrumento de variabilidade, dado que, através de ações incorretas, é possível que uma história atinja situações que dificilmente seriam alcançadas por personagens oniscientes.

Outra abordagem que tem sido proposta na tentativa de tornar os personagens mais críveis, porém pouco usada, é a modelagem de percepções [30, 29]. A utilização deste artifício tem por objetivo a geração de histórias sem a necessidade de personagens oniscientes, que sejam capazes de executar ações que estejam de acordo apenas com sua visão parcial de mundo. Uma das principais consequências desta simulação é a execução de ações incorretas, porém coerentes, visto que uma observação parcial do mundo faz do conhecimento dos personagens algo incompleto ou mesmo errôneo.

1.2 Proposição

Este trabalho propõe um método de simular percepções que permita aos personagens observarem e interpretarem erroneamente o ambiente e agirem de forma coerente com os resultados do processo. Como base para a utilização da percepção, também é proposto um modelo para a descrição de personagens e objetos que compõem mundos virtuais usados em sistemas de storytelling.

Para esta simulação, foram adotados como objetivos específicos:

- O desenvolvimento de um sistema de geração de histórias em turnos, que seleciona, para um personagem por vez, a ação que ele irá executar.
- A descrição dos personagens e objetos em termos de propriedades e características físicas.
- A manutenção de bases de conhecimento independentes para cada personagem com as informações que ele possui sobre o mundo, seu próprio conjunto de ações e seu próprio conjunto de regras de inferência de objetivos.
- O desenvolvimento de eventos que possam ter como efeito a modificação da descrição física de personagens e objetos.
- O desenvolvimento de métodos para mapeamento dos conhecimentos dos personagens em problemas e domínios de planejamento em PDDL.
- A criação de um método de atualização dos conhecimentos dos personagens que envolva uma análise do ambiente propensa a erros.
- A criação de métodos de tratamento de ações incorretas que mantenham a análise das ações e eventos da história, por parte dos personagens, coerente com os seus conhecimentos e lhes permitam aprender com seus erros ou prosseguir com a história acreditando estarem corretos.

1.3 Contribuições

As seguintes contribuições são resultado desta pesquisa:

- Um modelo para a descrição de personagens e objetos baseado em suas características físicas;

- Um processo de percepção dividido em uma etapa de observação (distingue o que os personagens são capazes de perceber) e uma etapa de interpretação (determina as expectativas de um personagem a respeito de elementos percebidos);
- Um tratamento de erros para a análise e execução dos efeitos de ações realizadas com base em informações erradas.

É importante deixar claro que esta pesquisa não propõe novos métodos de planejamento que envolvam ambientes parcialmente observáveis, mas sim a utilização de um modelo de percepção que permita a inserção proposital de informações incorretas na base de conhecimentos dos personagens para que, utilizadas em conjunto com algoritmos de planejamento existentes, sejam responsáveis pela geração de planos com ações incorretas, no entanto coerentes, e que venham a enriquecer a performance de personagens agentes no contexto de storytelling.

1.4 Estrutura

O capítulo 2 apresenta a fundamentação teórica necessária para o entendimento da utilização de técnicas de planejamento direcionadas ao contexto de storytelling, bem como uma análise de pesquisas da área com esse foco. O capítulo 3 detalha o trabalho inicial desenvolvido no decorrer desta pesquisa, no qual foi criado um sistema que se utiliza da inserção direta de erros nos conhecimentos que os personagens possuem, os testes realizados e os resultados obtidos. O capítulo 4 complementa o modelo apresentado no capítulo 3 com a inserção de extensões, como a possibilidade de execução de eventos, a simulação de percepção, a simulação de comportamentos e os resultados obtidos com o novo modelo. Por fim, o capítulo 5 apresenta as conclusões obtidas, discutindo questões envolvidas na modelagem de percepções desta pesquisa, as contribuições alcançadas e sugestões para possíveis trabalhos futuros.

Capítulo 2

Fundamentação Teórica

Neste capítulo são apresentados conceitos relacionados à geração dinâmica de histórias utilizados no desenvolvimento desta pesquisa. É detalhado como representações, conceitos e procedimentos envolvidos no estudo de planejamento são aplicados e direcionados ao processo de criação de histórias em sistemas de storytelling. São apresentados exemplos de sistemas de storytelling e, como neste trabalho é proposta uma simulação que visa orientar o processo de planejamento, por fim são revisadas estratégias relacionadas que vêm sendo utilizadas na área com o objetivo de direcionar o processo de planejamento à tarefa de criar histórias interessantes.

2.1 Planejamento e Representação de Problemas

Planejamento automático, ou simplesmente planejamento, é a subárea da inteligência artificial que estuda o processo computacional de deliberação que escolhe e organiza ações, antecipando seus efeitos na tentativa de alcançar metas pré-determinadas [11]. Seu estudo tem como principal resultado a construção de planejadores, processos responsáveis pela elaboração de métodos (ou planos), que possam ser aplicados ao estado inicial de um sistema de forma a conduzí-lo a um estado que se tem como meta.

Embora os planejadores tenham sido adaptados em diversas áreas para a resolução de problemas de domínios específicos, a área de planejamento automático procura desenvolvê-los para a aplicação de forma independente do domínio. Tendo em vista as dificuldades e necessidades específicas de cada problema, diversos modelos de planejamento têm sido desenvolvidos em conjunto com definições de domínios e problemas que representem suas características.

O mais simples destes modelos, normalmente chamado de planejamento clássico, define o domínio de um problema como uma tripla $\Sigma = \{S, A, \gamma\}$ na qual: S representa um conjunto de estados possíveis, A representa um conjunto de ações e γ representa uma função de transição de estados que associa estados a ações, definindo quais estados são alcançados pela aplicação de ações a determinados estados do domínio, isto é, se $\gamma(s, a) = s'$ então a aplicação da ação a ao sistema no estado s leva-o ao estado s'. A definição do problema, por sua vez, é feita como uma tripla $P = \{\Sigma, s, g\}$ onde Σ representa o domínio, s representa o estado inicial e g representa um conjunto de estados objetivo.

Em [11], Gallab apresenta um domínio formado por um contêiner posicionado em uma pilha, um robô capaz de se deslocar entre lugares vizinhos carregando um contêiner e um guindaste capaz de abastecer um robô com o contêiner que estiver no topo de uma pilha, chamado de domínio das docas (reproduzido na Figura 2.1). Também é apresentado um problema que diz respeito ao reposicionamento do contêiner que está na pilha do local 1 (situação representada na Figura pelo estado $s0$) para o local 2 (situação representada pelo estado $s5$). As Figuras 2.2 e 2.3 abaixo apresentam um exemplo de especificação do domínio e um exemplo de especificação do problema, respectivamente.

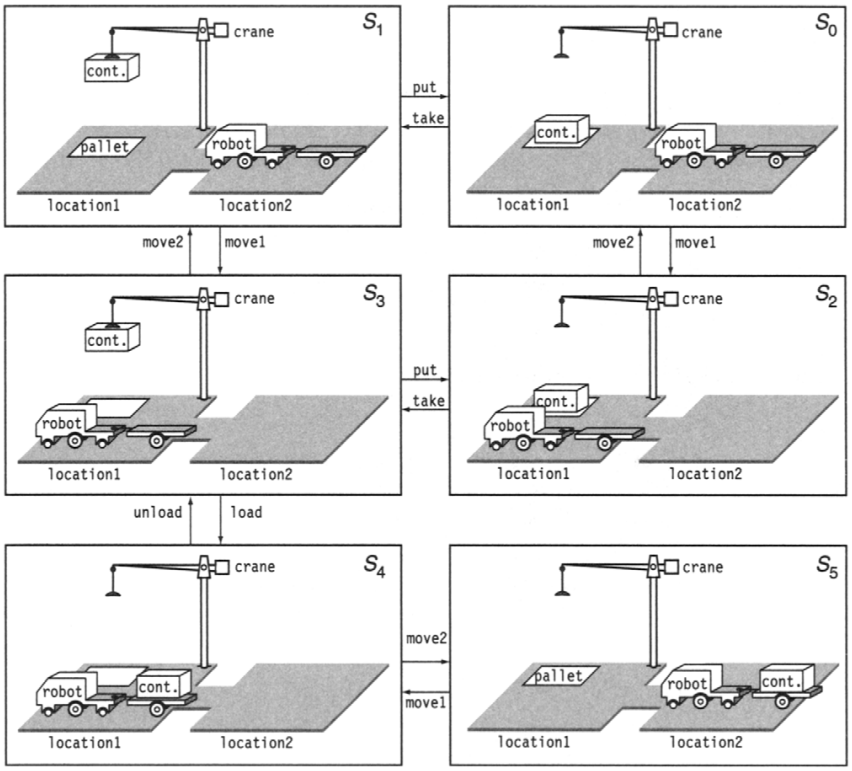

Figura 2.1: Domínio das docas

$\Sigma = \{S, A, \gamma\}$
$S = \{s0, s1, s2, s3, s4, s5, s6\}$
$A = \{take, put, load, unload, move1, move2\}$
$\gamma(S, A) = \{$

$\gamma(s0, take) = s1$	$\gamma(s0, move1) = s2$
$\gamma(s1, put) = s0$	$\gamma(s1, move1) = s3$
$\gamma(s2, take) = s3$	$\gamma(s2, move2) = s0$
$\gamma(s3, move2) = s1$	$\gamma(s3, put) = s2$
$\gamma(s3, load) = s4$	$\gamma(s4, unload) = s3$
$\gamma(s4, move2) = s5$	$\gamma(s5, move1) = s4\}$

Figura 2.2: Exemplo de especificação do domínio das docas.

$P = \{\Sigma, s0, g\}$
$g = \{s5\}$

Figura 2.3: Exemplo de especificação do problema.

Dadas estas especificações, a responsabilidade de um planejador passa a ser encontrar uma ou mais sequências de ações que, aplicadas ordenadamente ao estado $s0$, levem o sistema ao estado $s5$. Um exemplo de solução para este problema é o plano $\{move1, take, load, move2\}$.

O planejamento clássico se destina à resolução de problemas nos quais o conjunto de estados seja finito, o planejador tenha conhecimento completo acerca de cada estado do domínio, a aplicação de uma ação a um estado tenha como resultado um único estado, o sistema não possa sofrer alterações de estados que não pela aplicação de ações, o objetivo possa ser representado unicamente como um conjunto de estados, um plano seja uma sequência finita de ações ordenada linearmente, as ações tenham tempo implícito, isto é, não tenham duração e o sistema não passe por alterações de estado durante o processo de planejamento.

Embora estas restrições limitem consideravelmente o escopo de problemas que possam ser resolvidos através de planejadores clássicos, problemas que inicialmente não se enquadram nesse contexto podem ser adaptados ou divididos em descrições que sigam estas limitações, fato que tem facilitado o desenvolvimento de algoritmos de planejamento por reduzirem a necessidade de desenvolvimento de técnicas específicas para tratar problemas diferenciados.

As limitações também servem de base para a categorização de problemas e planejadores, pois as dificuldades e características de cada problema representam a necessidade

de relaxamento de uma ou mais restrições, o que determina qual tipo de problema está sendo tratado, como ele deve ser representado e quais técnicas de planejamento devem ser usadas ou precisam ser desenvolvidas para resolvê-lo. Entretanto, mesmo problemas de planejamento clássico podem apresentar certa dificuldade de resolução e exigir formas diferentes de representação.

A descrição de problemas seguindo o formato apresentado no exemplo das docas requer a enumeração de todos os estados e transições possíveis do domínio, o que, caso seja usado em um problema com uma grande quantidade de estados, pode se tornar uma tarefa impraticável. Como alternativas para essa descrição, Ghallab [11] detalha três representações, dentre as quais a denominada *representação clássica* é citada como a mais popular e utilizada neste trabalho.

A representação clássica se utiliza da lógica de predicados de primeira ordem para a descrição dos estados e ações. Cada estado é representado por um conjunto de instâncias de átomos que descrevem as propriedades de cada elemento envolvido no problema. No caso do domínio das docas, duas propriedades importantes para a definição de um estado dizem respeito ao posicionamento de um robô e de seu carregamento, que podem ser descritos respectivamente através dos predicados: $at(l, r)$ e $loaded(r, c)$. No estado $s5$, por exemplo, eles são instanciados com $at(location2, robot)$ e $loaded(robot, container)$. A representação completa de um estado envolve a descrição de cada propriedade relevante ao problema por meio de seus predicados e funções representativas. O domínio determina todos os estados possíveis não pela sua enumeração, mas sim pela definição de todos os predicados e funções necessários para a descrição do problema.

Seguindo a representação clássica, as ações devem ser representadas através de *operadores de planejamento* definidos por meio da quádrupla $O = \{n, param, precond, effect\}$ onde: n é um símbolo que referencia o operador (um nome), $param$ é um conjunto que descreve as variáveis que estão envolvidas no operador, $precond$ é uma conjunção de fórmulas atômicas que delimita dentro de quais situações o operador pode ser aplicado e $effect$ é uma conjunção de fórmulas atômicas que especifica quais mudanças são realizadas pela aplicação do operador a um estado. Nesse contexto, ações são instâncias de operadores de planejamento. No caso do domínio das docas, o operador responsável pela ação de movimentação do robô e um exemplo de instanciação podem ser descritos com a representação clássica, como se segue:

$operator\ \ move$
$param : (r, from, to)$

$precond : adjacent(from, to), at(r, from), \neg occupied(to)$
$effect : at(r, to), occupied(to), \neg occupied(from), \neg at(r, from)$

$action\ \ move$
$param : (robot, location1, location2)$
$precond : adjacent(location1, location2), at(robot, location1), \neg occupied(location2)$
$effect : at(robot, location2), occupied(location2), \neg occupied(location1), \neg at(robot, location1)$

Neste caso, a ação de movimentação tem como precondições os parâmetros to e $from$ serem adjacentes, o parâmetro r estar em $from$ e o parâmetro to não estar ocupado, e tem como efeitos o parâmetro r deixar de estar em $from$ para estar em to, o parâmetro to passar a estar ocupado e o parâmetro $from$ deixar de estar ocupado. Cada predicado estabelecendo uma característica para um parâmetro ou uma relação entre dois ou mais parâmetros.

A definição de problema segue as mudanças na representação dos estados, sendo formada pelo grupo de instâncias de átomos que caracterizam os estados inicial e objetivo.

Por se tratar de uma representação restritiva, foram desenvolvidas extensões à sintaxe que facilitassem o entendimento e diminuíssem o esforço necessários para a representação de domínios e problemas complexos [11]. Nas Figuras 2.4 e 2.5 são formalizados, respectivamente, parte do domínio e o problema mostrados anteriormente, através da representação clássica escrita em PDDL [10]. PDDL é uma linguagem de definição de problemas de planejamento, que foi criada no intuito de ser usada como linguagem padrão para a *International Planning Competition* - IPC e a cada edição vem recebendo novas extensões.

Duas das extensões são utilizadas nestas figuras, refletindo algumas das melhorias citadas à representação clássica. A primeira delas diz respeito à representação do objetivo do problema não através de instâncias de átomos que caracterizem conjuntos de estados objetivo completos, mas através de instâncias de átomos que se deseja alcançar, de tal forma que qualquer estado que esteja de acordo com estes literais é considerado um estado objetivo. A segunda delas diz respeito à organização dos elementos que compõem o problema em tipos, tais como robô, pilha e contêiner, que podem ser agrupados segundo uma hierarquia. A definição de tipos é feita através do uso de predicados unários que não mudam de estado para estado.

Como na Figura 2.4, a descrição do domínio é feita através da identificação dos tipos

de elementos envolvidos no cenário do problema, que nesse caso se resumem a *robot*, *location*, *pile*, *crane* e *container*; das propriedades que determinam as características de cada tipo e relações entre eles através de predicados como *at*, para definir a localização de um robô e *loaded* para definir se um robô está carregado com outro objeto; e das ações que representam as mudanças de estado, como a ação *move* que dissocia um robô da sua localização corrente e associa a uma localização adjacente. Na Figura 2.5, o problema é descrito pela instanciação dos elementos, nesse caso um robô, duas localizações, um contêiner, uma pilha e um guindaste; pela determinação das características de cada um dos objetos no estado inicial com instâncias dos predicados, como a adjacência das duas localizações, a posição inicial do robô e a posição inicial do contêiner; e por fim pela determinação das características que se deseja alcançar: a posição final do robô, que deve estar carregado com o contêiner.

```
(define (domain docks)
    (: types location pile robot crane container)
    (: predicates
        (adjacent ?l1 ?l2 - location)
        (attached ?p - pile ?l - location)
        (at ?r - robot ?l - location)
        (occupied ?l - location)
        (loaded ?r - robot ?c - container)
        (holding ?k - crane ?c - container)
        ...
    )

    (: action move
        : parameters (?r - robot ?from ?to - location)
        : precondition (and (adjacent ?from ?to) (at ?r ?from) (not (occupied ?to)))
        : effect (and (at ?r ?to) (not (occupied ?from)) (occupied ?to) (not (at ?r ?from)))
    )
    (: action load
        : parameters (?k - crane ?c - container ?r - robot ?l - location)
        : precondition (and (at ?r ?l) (belong ?k ?l) (holding ?k ?c) (unloaded ?r))
        : effect (and (loaded ?r ?c) (not (unloaded ?r)) (empty ?k) (not (holding ?k ?c)))
    )
)
```

Figura 2.4: Especificação de parte do domínio das docas em representação clássica, conforme [11].

Neste exemplo, a representação aparenta exigir maior esforço que o modelo apresen-

```
(define (problem docks1)
    (: domain docks)
    (: objects location1 location2 - location pile1 - pile robot1 - robot
                crane1 - crane container1 - container)
    (: init
        (adjacent location1 location2)
        (adjacent location2 location1)
        (at robot1 location1)
        (unloaded robot1)
        (empty crane1)
        (in container1 pile1)
        (top container1 pile1)
    )
    (: goal (and (at robot1 location1) (loaded robot1 container1)))
)
```

Figura 2.5: Especificação do problema.

tado inicialmente, pois existem mais características do que estados. Porém, caso fossem adicionadas apenas duas localizações vazias ao problema, o modelo inicial exigiria a enumeração do dobro de estados, enquanto que o exemplo de representação clássica necessitaria apenas da adição das novas localizações aos objetos definidos na representação do problema e das adjacências dessas localizações, sem alterações na especificação do domínio.

2.2 Geração e Representação de Mundo

Definida a utilidade dos algoritmos de planejamento e o contexto no qual se inserem, o processo pelo qual histórias podem ser geradas computacionalmente através de planejadores torna-se evidente. Uma história é formada por uma série de ações ordenadas logicamente alimentadas por fatores do mundo onde ela se passa e pelos personagens que habitam nele. Portanto, é possível usar um planejador para gerar uma sequência de ações que caracterize uma história, partindo da descrição das características e leis de um mundo virtual como um domínio de um problema, e das situações inicial e final como um problema de planejamento.

A delimitação das características, regras e personagens do mundo virtual vem sendo realizada de diversas formas pelos sistemas de storytelling, na busca pela geração de histórias interessantes com necessidades diferentes e partindo de premissas diferentes.

O sistema de storytelling TaleSpin [17], considerado o pioneiro da área, trabalha com uma representação de mundo formada por personagens, objetos e cenários. Os personagens são o elemento principal do processo de criação usado pelo sistema, sendo representados por meio de características próprias, tendo cada um sua própria interpretação de mundo e conjunto de ações que sabe realizar. O processo de geração ocorre pela atribuição de uma necessidade a um personagem tal como fome ou sede, usada pelo sistema para lhe atribuir um objetivo que junto a sua visão de mundo formam um problema de planejamento. As ações realizadas pelo personagem na busca pela resolução deste problema compõem a história criada pelo sistema. Algumas destas ações podem incluir a interação com outros personagens, o que pode resultar em novos objetivos para estes personagens que também passam a atuar. A abordagem na qual os personagens têm uma representação interna de mundo e as ações das histórias são resultado destes personagens seguindo seus próprios objetivos é classificada pela literatura como baseada em personagens [5, 13, 29].

Por outro lado, o sistema Universe [14] se utiliza de uma base de dados de personagens descritos também por meio de características próprias, e de uma base de dados de eventos. Dentro da área de planejamento automático, eventos são mudanças de estado que não são resultado direto da ação do planejador, mas sim da própria natureza do ambiente. No contexto de storytelling, onde o mundo no qual a história ocorre é simulado pelo sistema, eventos são "ações" do mundo usadas para simular mudanças que podem envolver os personagens, mas não são controladas diretamente por eles. O sistema Universe se destina a geração de estruturas básicas de história como ferramenta de auxílio a autores. O sistema recebe do usuário uma especificação de situações que deseja ver incluídas na história final e executa o processo de geração usando essas situações como objetivos e os eventos como operadores. Estas especificações predeterminadas têm sido chamadas de objetivos do autor e a abordagem que tem estes objetivos como guia para o processo de geração é classificada como baseada em enredo [27, 23, 9].

O motivo que levou à criação destas duas abordagens foi a preocupação dos sistemas de storytelling com a liberdade e a qualidade das histórias geradas. No caso da abordagem baseada em personagens, como eles são guiados pelos seus próprios objetivos e não são forçados a seguir um roteiro predefinido, também são capazes de atuar com maior liberdade. Enquanto um personagem busca seus próprios objetivos, suas interações com os demais personagens e com os demais elementos do mundo podem levá-lo a desenvolver situações interessantes não previstas. O chamado *character-based storytelling* ou simplesmente storytelling baseado em personagens visa o desenvolvimento de histórias interessantes através da simulação do comportamento de personagens em um mundo

virtual que sejam capazes de gerar situações inusitadas porém cativantes. Geralmente, essa simulação é feita através do uso de um agente para cada personagem, representado através das estruturas do próprio personagem ou por meio de estruturas independentes com características próprias. Neste último caso os agentes atuam levando em conta suas próprias características e conhecimentos, e têm sido chamados de atores virtuais [29].

As desvantagens relacionadas ao uso desta abordagem são também consequência da sua característica principal: o desenvolvimento de situações interessantes é uma **possibilidade** gerada pela liberdade dada aos personagens. Não necessariamente esta simulação será capaz de gerar histórias consistentes pois não há um compromisso estrito com o seu desenvolvimento e, já que todo o processo se baseia nos personagens, é necessário que eles sejam descritos com maior nível de detalhe e os algoritmos de planejamento tentem reproduzir o comportamento humano, o que, feito com maior ou menor nível de realismo pode comprometer o desempenho do sistema. No quesito interatividade, a abordagem mostra-se mais promissora devido à própria capacidade de adaptação: como os personagens não são forçados a seguir um roteiro, suas respostas em um sistema de storytelling interativo às influências de um usuário são mais naturais e o mundo virtual tem mais liberdade para demonstrar ao usuário as consequências de suas escolhas.

A abordagem baseada em enredo, conhecida na literatura como *plot-based storytelling*, visa uma geração mais comprometida com a qualidade e, para isso, dedica-se a seguir os objetivos pré-estabelecidos guiando os eventos do mundo e/ou as ações dos personagens de forma a garantir que eles sejam atingidos. Sistemas de storytelling com geração baseada em enredo podem ou não simular o comportamento de personagens através de agentes. Quando o fazem, atribuem-se aos personagens objetivos que os levam a executar ações que possam conduzir a história por entre as situações esperadas. De outra forma, histórias podem ser geradas segundo essa abordagem sem que os personagens tenham uma representação interna do mundo, a partir de eventos que englobem suas ações. Este caso é similar a tratar o próprio mundo virtual como um personagem agente no qual os eventos são considerados suas ações, os personagens habitantes são considerados suas características e os objetivos do autor são considerados seus próprios objetivos. A proposta desta abordagem confere ao sistema mais chances de se gerar histórias interessantes.

A desvantagem de se usar a abordagem baseada em enredo está na liberdade limitada dada aos personagens. A obrigação de seguir um roteiro limita suas ações àquelas que levam a história aos pontos desejados, além de poder tornar suas atitudes irreais. A necessidade de uma história chegar a determinados pontos pode forçar os personagens

a tomar atitudes que são contra sua natureza. Voltando ao exemplo do herói que deve salvar a princesa, se for determinado que o vilão deve estar morto ao final da história, o herói terminará por matá-lo se não houver outras opções de ações ou eventos que levem ao final, apenas para que o final seja alcançado, mesmo que a ação não esteja de acordo com a suposta índole ou atributos do personagem. Em termos de interatividade, um usuário de um sistema de storytelling baseado em enredo interativo pode, por exemplo, após criar o roteiro, definir quais personagens farão parte da história ou ainda por que meios o roteiro será seguido (quais ações serão executadas ou quais dentre suas opções de efeitos serão aplicados).

Na literatura é possível encontrar diversas pesquisas relacionadas ao desenvolvimento de sistemas de ambas as abordagens que geram histórias, inclusive com simulações mais realistas de personagens e mundos virtuais através do relaxamento das restrições relacionadas ao planejamento clássico, como é o caso do trabalho de Porteous em [22], onde admite-se a utilização de ações com tempo explícito, chamadas ações durativas. Para tal, diferentes algoritmos de planejamento e representações de problemas são utilizados. A seguir são apresentados exemplos de sistemas de geração de histórias baseados em personagens, em enredo e em uma abordagem híbrida, respectivamente, bem como as técnicas de planejamento que foram empregadas.

2.2.1 Character-based Storytelling

Em [5], Cavazza apresenta um sistema de storytelling com geração baseada em personagens, cujo objetivo é permitir ao usuário a possibilidade de interagir com os personagens e o ambiente a qualquer momento da apresentação da história, de forma a influenciar o seu desenvolvimento. O cenário base utilizado como exemplo é o de um episódio de uma popular série de televisão chamada Friends, onde o personagem Ross tenta convidar a personagem Rachel para sair, e, para atingir seu objetivo, ele deve antes presenteá-la com algo que ela goste.

O processo de planejamento utilizado no trabalho de Cavazza se utiliza de uma estrutura denominada rede de tarefas hierárquica (*hierarchical task network* - HTN) para simular o processo de raciocínio dos personagens. Esse processo se diferencia do planejamento clássico devido à utilização dos conceitos de tarefa, rede de tarefas e método. Tarefas são formas de se resolver problemas HTN. Algumas delas são chamadas de não-primitivas e podem ser divididas em tarefas menores recursivamente até que se chegue a tarefas que não podem ser divididas, classificadas como tarefas primitivas. Métodos são

associados a tarefas e usados como forma de definição de suas possíveis decomposições. As tarefas derivadas de uma tarefa não-primitiva são organizadas por meio do uso de restrições que definem sobre quais condições elas podem ser executadas bem como a ordem em que elas devem ser executadas. Tarefas derivadas são ligadas à tarefa superior por meio de arestas. Tarefas não-primitivas não podem ser derivadas em novas tarefas, ao invés disso, elas são cumpridas por ações pré-estabelecidas de mesmo nome e que possam ser executadas nos estados onde a tarefa é aplicada. O conjunto de tarefas interligadas por meio de arestas, restrições de aplicabilidade e restrições de ordenação é chamado de rede de tarefas. Uma mesma rede de tarefas pode instanciar diversas formas de se resolver o mesmo problema.

A Figura 2.6, reproduzida do trabalho de Cavazza [5], detalha um exemplo de parte da rede de tarefas utilizada pelo personagem Ross, onde as extremidade representam tarefas primitivas e as demais tarefas não-primitivas. A tarefa principal do personagem é chamar a personagem Rachel para sair. Essa tarefa se divide nas subtarefas *acquire information*, *gain affection*, *isolate her* e *ask her*. No caso deste trabalho, a tarefa *acquire information* é dividida em outras três subtarefas que são usadas como opções. O personagem Ross deve tentar descobrir qual presente é o mais apropriado e para isso conhece estas três subtarefas como forma de obter essa informação. Cada uma destas subtarefas, por sua vez, é dividida nas tarefas primitivas que serão de fato executadas pelo personagem. Dessa forma, as tarefas *acquire information*, *borrow her diary* e suas tarefas primitivas constituem um método para a resolução da rede apresentada na Figura.

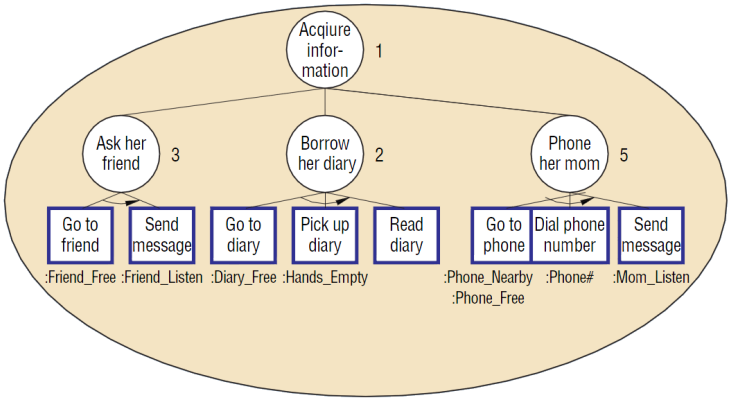

Figura 2.6: Exemplo de HTN

Cada um dos personagens possui sua própria HTN e consequentemente um papel na história. O sistema trabalha apresentando o resultado das ações dos personagens em um ambiente 3D, que é utilizado pelo usuário como meio de interação. A definição de quais tarefas serão utilizadas constitui o processo de geração de histórias que pode ser

influenciado pelo usuário do sistema. Como cada tarefa possui suas próprias precondições para ser realizada, como por exemplo, o personagem Ross só pode pegar o diário se ele estiver em sua posição inicial. Dessa forma, é possível definir o desenvolvimento da história através de mudanças no cenário que venham a possibilitar ou impossibilitar certas tarefas. Ao usuário é dada a chance de mudar posições de itens chave como é o caso do diário. Também é possível passar informações diretamente aos personagens, como por exemplo dizer ao Ross qual seria o presente ideal para a Rachel, de forma a influenciar na escolha de qual tarefa realizar na HTN referente a subtarefa *gain affection*. A escolha de quais tarefas são executadas por Ross também podem afetar por exemplo o humor dos demais personagens ou resultar em situações inusitadas, como um encontro prematuro entre Rachel e Ross que só deveria acontecer na etapa final da história.

O sistema consegue então extrair variabilidade a partir do posicionamento inicial dos personagens da história, das interações com o usuário e das opções de tarefas que são dadas aos personagens, que, por conseguinte, podem causar mudanças de humor e a emergência de novas situações que resultem em novas possibilidades de variação.

Relacionando esse trabalho ao que apresentamos nesta dissertação, é possível notar que o sistema proposto por Cavazza também se utiliza de ações incorretas para o enriquecimento da história. O personagem Ross não possui inicialmente a informação do que dar como presente e a partir do que lhe for dito pelo usuário ou pelos demais personagens, ele pode vir a presentear Rachel com algo que ela não queira, o que mesmo estando de acordo com o conhecimento que ele tem sobre o mundo, vai contra o objetivo inicial do personagem. Neste caso, no entanto, as ações incorretas são resultado das opções que são dadas aos personagens, não sendo possível a realização de tipos diferentes de erros que não os pré-estabelecidos pela rede de tarefas. O que o sistema possibilita ao personagem é acreditar que certas precondições estejam corretas quando na verdade estão erradas.

2.2.2 Plot-based Storytelling

Vladimir Propp em [24] faz uma análise sobre contos de fadas russos e detalha a existência de padrões nos acontecimentos que compõem todas essas histórias, tais como raptos de vítimas por vilões ou lutas entre vilões e heróis. Foram identificados 31 padrões cujas combinações seriam capazes de gerar qualquer um dos contos de fadas estudados ou ainda novas histórias do mesmo gênero.

Partindo de uma formalização destes acontecimentos como eventos através da especificação de pré-condições e efeitos, o sistema de storytelling interativo Logtell [23, 7] é um

exemplo de sistema com geração baseada em enredo que se propõe a gerar e dramatizar histórias interessantes tendo por base os padrões identificados por Propp. A interatividade proposta pelo sistema permite ao usuário influenciar o desenvolvimento das histórias a partir da definição do estado inicial e dos estados que ele deseja ver concretizados.

O processo de geração de histórias empregado pelo Logtell se utiliza de um planejador não-linear, que é capaz de gerar planos que não exijam uma ordenação completa dos eventos que compõem a história, denominado IPG [23]. Em suma, a resolução de determinados problemas de planejamento não exige planos que garantam uma ordenação estrita entre as ações que os compõem, sendo possível que algumas de suas ações sejam executadas na ordem que for conveniente ou mesmo em paralelo, desde que sejam executadas. Planejadores não-lineares se propõem a encontrar planos sem necessariamente impôr uma ordem estrita de execução. O IPG é então capaz de criar histórias cujos eventos estejam parcialmente interligados e cuja ordem possa ser definida posteriormente.

Assim como no caso do sistema Universe, o Logtell gera histórias a partir da instanciação e ordenação de eventos, nesse caso característicos de um gênero específico e que representam situações envolvendo a atuação de personagens. Também são utilizadas regras de inferência de objetivos para promover a escolha de sequências de eventos características do gênero. Considere-se, por exemplo, o evento "ato de vilania" caracterizado por Propp como um vilão que causa algum dano a uma vítima. Uma das regras de inferência de objetivos usadas no sistema diz que se no início da história houver uma vítima, ela executará uma ação que a tornará vulnerável. Essa regra é utilizada no sistema para guiar o planejador a escolher um evento que torne a vítima vulnerável, permitindo a futura escolha de um evento que represente o ato de vilania, uma sequência comum em contos de fadas.

O sistema alterna sua execução entre etapas de interação, planejamento e dramatização. As etapas de interação e planejamento ocorrem em um primeiro momento onde o sistema fornece ao usuário um esquema de uma história com o qual ele pode interagir através de modificações, que são analisadas pelo planejador para verificação de incoerências. A fase se repete até que seja criada uma história coerente que satisfaça o usuário e com ela o sistema prossegue para a dramatização. A dramatização é feita através de personagens reativos que seguem o roteiro pré-estabelecido pelos eventos. O sistema vem recebendo diversas extensões [15, 12, 27] e dentre elas, o trabalho de Silva [27], que inseriu a possibilidade de interação durante a etapa de dramatização pelo uso de ações que não possuam um efeito exato gerenciadas por um novo planejador HTN não determinístico.

As Figuras 2.7 e 2.8, reproduzidas respectivamente de [23] e [15], apresentam a interface de interação do sistema com o usuário e um exemplo da etapa de dramatização.

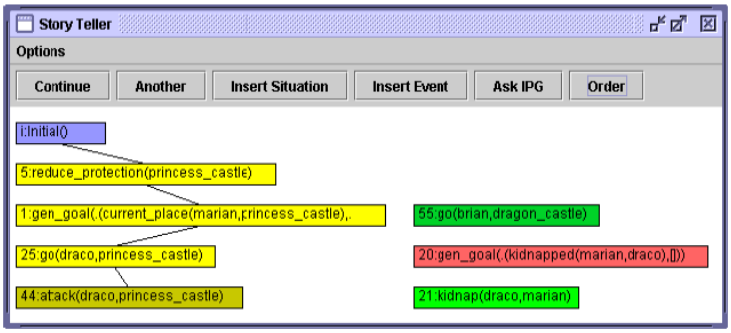

Figura 2.7: Interface do Logtell para a geração de histórias

Figura 2.8: Exemplo de dramatização

No caso desse sistema, mesmo que as atitudes dos personagens possam ser interpretadas como incorretas, como seria o caso da vítima executando uma ação que a torne indefesa, elas não seguem a mesma proposta da presente pesquisa, já que não são um efeito colateral viabilizado por meio de informações ou raciocínio incorreto, isto é, não são erros que podem ser gerados pelo processo de planejamento e sim que foram predefinidos pela base de dados do sistema. A abordagem baseada em enredos visa garantir que situações predefinidas pelo usuário sejam devidamente alcançadas, o que limita o uso de ações incorretas geradas dinamicamente pois elas seriam admitidas apenas quando não inviabilizassem o script.

Em [9], Dória apresenta a utilização de autômatos não determinísticos na etapa de dramatização do sistema Logtell para garantir variabilidade na apresentação dos eventos de uma mesma história. Como um mesmo evento pode ser dramatizado de diversas

formas (apresentado por sequências de ações diferentes), neste trabalho são usados autômatos para descrever eventos por meio de diferentes ciclos de ações (que podem incluir ações incorretas) controlados de forma a sempre caracterizar as precondições e efeitos dos eventos descritos.

2.2.3 Abordagem Híbrida

No intuito de encontrar um equilíbrio entre o nível de liberdade dos personagens e a qualidade do desenvolvimento de histórias, algumas pesquisas têm se dedicado a utilizar características de ambas as abordagens criando métodos híbridos. Este é o caso do sistema de storytelling interativo Façade [16] no qual os personagens têm objetivos próprios, mas atuam sob influência de um módulo externo denominado *drama manager*.

O cenário exemplo desenvolvido pelos autores do sistema se passa no apartamento de um casal que se encontra prestes a se separar. O objetivo do usuário é, assumindo o papel de um amigo, impedir que o divórcio aconteça. O sistema permite que o usuário interaja com o ambiente por meio da manipulação de objetos específicos e com os personagens através de linguagem natural. As ações executadas pelo usuário, bem como o diálogo com os personagens influencia o comportamento deles durante o desenvolvimento da história e determina a conclusão. A Figura 2.9 apresenta um exemplo de execução do cenário criado pelo sistema com texto escrito pelo usuário.

Figura 2.9: Exemplo de execução do Façade

A geração de histórias neste sistema se baseia em dois elementos característicos das demais abordagens a simulação de comportamentos e o *drama manager*. A simulação de comportamentos é feita pela atribuição e ordenação de objetivos aos personagens quando

determinadas situações são alcançadas, isto é, são conjuntos de regras associadas a objetivos que são parte da definição dos personagens e quando atendidas lhes atribuem os objetivos correspondentes. A simulação também engloba a manipulação de objetivos por meio de regras que determinam sua ordem de prioridade e o critério de abandono. No caso deste sistema, os personagens atuam por meio de *beats*, definidos pelos autores como conjuntos de comportamentos e que possuem suas próprias precondições e efeitos.

O exemplo foi desenvolvido de tal forma que múltiplos *beats* possam ser atendidos em uma mesma situação e o sistema seja capaz de escolher aqueles que forem mais apropriados para que a história tenha uma boa qualidade. De acordo com Swartjes em [29], para que uma sequência de ações possa ser vista como uma narrativa, é necessário que elas possam ser delimitadas em momentos que representem introdução, desenvolvimento e conclusão. No decorrer de uma história, os personagens saem de um momento de estabilidade e eventualmente enfrentam obstáculos que elevam a tensão da história, enquanto que a resolução destes obstáculos reduz a tensão. Esta continua com uma curva crescente até que os personagens enfrentem o maior obstáculo e o resolvam, levando a tensão novamente aos níveis iniciais.

O sistema de Façade guia o processo de geração de histórias atribuindo, aos efeitos dos *beats*, valores de variação de tensão que são usados pelo *drama manager* para escolher quais *beats* executar em que momentos da história de forma a causar as variações de tensão características de narrativas. O processo de geração visa construir o arco de tensão aristotélico apresentado na Figura 2.10.

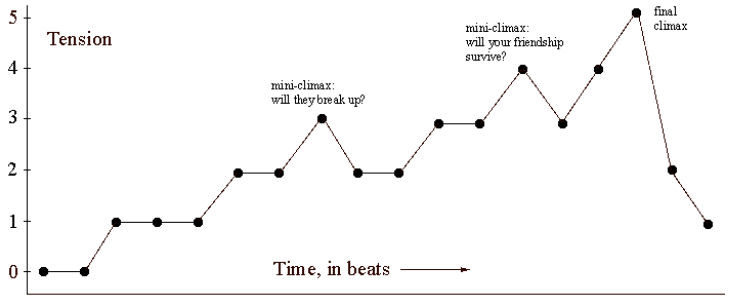

Figura 2.10: Curva de tensão aristotélica

Quanto ao uso de ações incorretas, essa abordagem híbrida em específico se assemelha ao caso da abordagem baseada em enredos, já que por serem utilizados *beats* predeterminados, não é possível a geração dinâmica de informações incorretas que possam vir a influenciar na escolha de ações. Além do fato de que os *beats* são escolhidos de forma a necessariamente gerar o arco de tensão desejado.

2.2.4 Orientando o Processo de Planejamento

Planejadores são desenvolvidos tendo como objetivo a resolução de problemas através da melhor solução que encontrarem, o que condiz com as necessidades dos problemas no contexto de Inteligência Artificial, mas que dificulta o processo de geração de histórias interessantes. Independente da abordagem usada, o processo de planejamento puramente aplicado no contexto de storytelling tem como resultado histórias curtas e personagens com atuações não naturais. Em vista deste inconveniente, técnicas como a simulação de arcos de tensão e simulação de comportamentos por meio de regras de inferência de objetivos têm sido estudadas e aplicadas nos sistemas de storytelling de forma a "guiar"seus processos de planejamento. No caso da utilização de arcos de tensão, por exemplo, dadas as opções de ação que um personagem possui em um momento específico de uma história, o processo de planejamento é orientado a escolher a ação cuja variação de tensão siga o esperado para a concretização do arco de tensão desejado. No contexto de planejamento, esta orientação pode ser feita através da adição de um valor de tensão ao domínio do problema, da inclusão de efeitos que alterem este valor de tensão nas ações e de objetivos relacionados à obtenção de valores específicos de tensão na especificação do problema.

Técnicas específicas da literatura e a simulação de processos mais detalhados que envolvem o raciocínio e o conhecimento humano têm sido a base para a orientação de planejadores pela criação de problemas e de domínios cujo objetivo não seja a resolução do problema em si, mas sim a obtenção de um plano que corresponda às ações que os personagens fariam para resolver o problema ou que corresponda à sequência de eventos que melhor caracterizem uma história de um determinado gênero.

Nos trabalhos de Damiano e Lombardo [8] e Lebowitz [14], o artifício usado pelos autores é a descrição dos personagens através de atributos que descrevam suas personalidades que, aplicados às precondições e aos efeitos de suas ações ou, no caso de Lebowitz, dos eventos da base de dados do sistema, resultem em histórias cujos personagens realizam apenas ações ou participam apenas de eventos que estejam de acordo com suas índoles.

Como já mencionado, o processo de planejamento pode ser usado em storytelling para simular o raciocínio de personagens através da descrição da sua visão de mundo como um problema de planejamento. No entanto em Aylett [2] e, Pizzi e Cavazza [20] o processo de planejamento também é usado para prever suas expectativas futuras. A partir do que um personagem sabe sobre o mundo e sobre os demais personagens, o sistema usa o planejador para criar uma sequência de ações que represente o que ele espera do futuro. Estas expectativas são usadas para causar alterações sobre valores

que representam emoções, que seguem a mesma ideia dos atributos citados no parágrafo anterior: determinadas ações só podem ser executadas quando determinadas restrições sobre as emoções são cumpridas. No trabalho de Aylett [2], por exemplo, foi desenvolvido um jogo interativo sobre o tema bullying onde o usuário deve dar dicas ao personagem vítima do problema sobre como ele deve agir. O personagem inicialmente evita executar a ação revidar pois suas expectativas futuras com relação ao resultado da ação envolvem problemas para si próprio, o que aumenta o seu medo e o impossibilita de realizar a ação.

Mieke Bal [3] propõe o estudo de uma história a partir de sua separação em três níveis: fábula, história e texto narrativo. A fábula é composta por todas as ações e eventos que se passam no mundo virtual em ordem cronológica, a história é formada pela seleção de ações e eventos específicos em uma ordem na qual possa despertar sentimentos como curiosidade e suspense e o texto narrativo representa o meio pelo qual a história é contada. Swartjes [30], propõe a geração de histórias baseado nesta divisão, especificando um modelo para a criação e representação de fábulas através de ações e eventos ligados entre si por meio de relações causais. Estas relações seriam usadas em uma etapa seguinte para a seleção de quais sequências de ações e eventos poderiam ser usadas na criação da história em si.

Na linha de abordagens híbridas, Riedl [25] propõe um processo de geração de histórias no qual os personagens executam ações em prol dos objetivos estabelecidos pelo autor, mas apenas aquelas que seriam executas mesmo se ele estivesse seguindo seus próprios objetivos. Isso é realizado através da criação de planos que sigam os objetivos dados pelo autor e planos que sigam os objetivos estabelecidos para os personagens, que passam a executar os planos gerados pelo primeiro caso, cujas ações estejam envolvidas nos planos gerados pelo segundo caso. O intuito dessa técnica é fazer com que os personagens alcancem o que foi predefinido pelo autor, sem que para isso pareçam estar agindo unicamente para o bem da história.

Capítulo 3

Planejamento com Ruído Dinâmico para Storytelling Emergente

Este capítulo detalha parte do trabalho desenvolvido no decorrer desta pesquisa cujo objetivo prático foi a criação de um processo de geração de histórias lançando mão da execução proposital de ações incorretas. Para tanto, este processo insere erros dinâmicos controlados, aqui chamados de ruídos, diretamente no conhecimento dos personagens. Ao utilizá-las em conjunto com planejadores, tem-se como resultado a geração de planos propositalmente incorretos, mas cujas ações não inviabilizam o desenvolvimento da história e lhe confere fatores não determinísticos.

Neste capítulo é apresentado e aplicado o conceito de ruído como forma de causar ações erradas, mas ainda sem um mecanismo que justifique a sua inserção do ponto de vista de um personagem. No Capítulo 4 os ruídos são desenvolvidos e adaptados à simulação de percepções nos métodos de observação e interpretação do mundo, o que lhes concede a coerência necessária para aplicação em storytelling.

Através da criação deste trabalho inicial foi possível realizar uma análise do potencial que ações incorretas possuem de influenciar uma história, das dificuldades envolvidas na necessidade de se controlar as consequências destas ações, assim como das justificativas necessárias para que elas possam ser utilizadas. As subseções seguintes apresentam a arquitetura do processo desenvolvido (seção 3.1) que envolve a descrição do fluxo do processo gerativo (seção 3.1.1) e a representação utilizada para a simulação de um mundo virtual como um sistema de estados (seção 3.1.2), a especificação dos ruídos e como eles são utilizados (seção 3.2), os testes realizados (seção 3.3) e uma breve discussão com a análise dos resultados (seção 3.4).

3.1 Arquitetura

Como este trabalho se propõe a utilizar o conhecimento que os personagens têm sobre o mundo, no objetivo de gerar ações incorretas e com isso variabilidade, a abordagem baseada em personagens foi adotada por se mostrar mais apropriada. Dessa forma, o processo de geração se utiliza da representação dos seus conhecimentos de mundo e de seus próprios objetivos como problemas e domínios de planejamento, para usá-los como insumos de planejadores e obter sequências de ações que representem as atuações dos personagens.

O trabalho não se propõe a criar novos algoritmos de planejamento, por esse motivo a representação é feita através da linguagem PDDL, para que seja compatível com planejadores já existentes. O processo de geração é responsável pela definição das ações que compõem a história, o que envolve não só a escolha destas ações como também a ordenação e o fluxo de sua execução, mais especificamente da passagem de tempo.

3.1.1 Linha do Tempo

Em sistemas baseados em personagens e mesmo em alguns sistemas baseados em enredo, onde os operadores representam as ações individuais de cada personagem, o uso de múltiplos personagens traz a necessidade de tratamento do ritmo no qual elas são executadas. Na literatura é possível encontrar diferentes formas com as quais os sistemas de storytelling têm tratado a passagem do tempo. Em Swartjes [29], por exemplo, o sistema divide o fluxo da história em rounds, dentro dos quais cada personagem observa o mundo (atualiza sua base de conhecimento), planeja uma sequência de ações e a executa, sendo possível que uma ação se estenda por mais de um round. Os rounds não são individuais, o que viabiliza aos personagens executarem ações ao mesmo tempo, mas, neste caso, sem interdependências, por não ser possível estabelecer efeitos específicos para o intervalo de execução de uma ação.

Já em Porteous [22], não só os personagens podem atuar ao mesmo tempo, como podem influenciar o resultado das ações uns dos outros e em tempo contínuo, o que é possibilitado pelo uso de ações que têm um tempo de duração explícito, e estabelecem mudanças nas condições do ambiente no início, no decorrer e no final de sua execução, chamadas de ações durativas. A execução de ações em paralelo traz consigo a necessidade de tratar problemas de concorrência, como seria o caso de, em uma determinada situação onde os personagens estejam competindo entre si, uma ação ser feita para que outra

seja interrompida. Para evitar o surgimento de tais situações e, por elas não serem estritamente necessárias para a demonstração das possibilidades de desenvolvimento de histórias viabilizadas pelo uso de ações incorretas, o sistema desenvolvido neste trabalho se utiliza de uma divisão da linha do tempo em turnos, com ações não-durativas.

O processo de geração por turnos opera dividindo a linha do tempo em ciclos, que por sua vez são divididos em turnos. Por ciclo, cada personagem recebe um turno com o qual ele atualiza sua base de conhecimentos, traça um plano que o leve ao seu objetivo e executa a primeira ação do plano criado. A execução de uma ação é feita aplicando-se seus efeitos ao estado corrente do mundo.

Os turnos são concedidos em uma sequência predefinida e as ações são executadas em série, portanto, dentro de um mesmo ciclo, cada personagem vê o mundo já com os efeitos das ações dos personagens que atuaram em turnos anteriores. Essa divisão da linha do tempo assemelha-se à utilizada em jogos de RPG, onde cada personagem jogador tem a "sua vez" de atuar dentro de cada ciclo. O algoritmo 3.1 apresenta como as ações são organizadas através de um *loop* principal que separa a linha do tempo em ciclos e se encerra quando nenhum personagem tiver atuado dentro de um ciclo, e de um *loop* interno que divide um ciclo em turnos e se encerra quando cada personagem tiver concluído seu turno. A Figura 3.1 mostra uma visão esquemática da mesma, com cada personagem atuando uma vez dentro de um mesmo ciclo, em uma ordem predefinida:

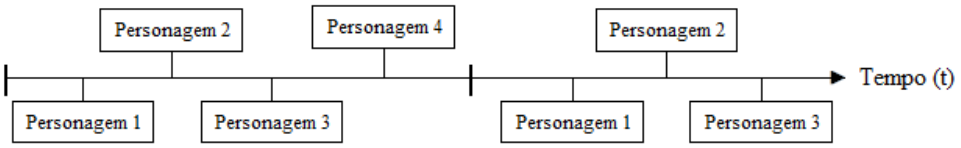

Figura 3.1: Linha do tempo em turnos

A geração de histórias dividindo a linha do tempo em turnos pode ser desvantajosa se utilizada com um número considerável de personagens, pois cada um deles volta a atuar apenas quando todos os demais já executaram suas ações, podendo acontecer do ambiente ter sido afetado consideravelmente, incluindo o próprio personagem que não é capaz de reagir sob a ação dos demais. Não é possível para este sistema, por exemplo, simular ações de combate com personagens bloqueando ou se desviando de ataques. Além disso, todas as ações são executadas como se durassem exatamente o mesmo tempo.

A etapa de planejamento executada neste trabalho é feita sem que um personagem considere as ações futuras dos demais, dessa forma sempre que o sistema for capaz de encontrar um plano de ações para um personagem, por ele não considerar a possibilidade

Algoritmo 3.1 Algoritmo de geração baseado em turnos.

Entrada: *mundo* contendo personagens, objetos e um mapa.
Saída: *história* contendo uma lista de ações.

```
personagens := lista com os personagens do mundo;
nPersonagens := quantidade de personagens do mundo;
enquanto nPersonagensIndecisos != nPersonagens faça {loop de uma história}
  {nPersonagensIndecisos conta o número de personagens inativos em um ciclo}
  nPersonagensIndecisos := 0;
  para todo personagem em personagens faça {loop de um ciclo}
    atualizaConhecimento(personagem, mundo);
    plano = planejarParaPersonagem(personagem, personagem.objetivo);
    {plano contém a sequência de ações que o personagem assume que o levará ao seu
    objetivo}
    se plano.tamanho != 0 então
      ação = primeira ação em plano;
      atualizaMundo(ação, mundo); {Aplica os efeitos da ação ao mundo}
      incrementaHistória(ação, história); {Adiciona a ação à história}
    senão
      incrementa nPersonagensIndecisos;
retorna (história);
```

das características do mundo se alterarem a seu favor em turnos posteriores por efeito das ações dos demais, o personagem necessariamente executará a primeira ação do plano. O que, dependendo dos objetivos de cada um, pode causar um *livelock* com os personagens competindo entre si, mas nunca alcançando suas metas. Caso fosse utilizado um planejamento com antecipação de ações, um personagem poderia prever a possibilidade de agir em um turno futuro onde as condições do mundo estivessem favoráveis ao seu objetivo.

Entretanto, é possível reduzir o impacto na autonomia dos personagens modelando-os com ações básicas que, em sua maioria, não tenham efeitos que causem mudanças consideráveis no ambiente, e utilizando poucos personagens (aplicando o sistema de turnos apenas aos personagens principais, com os coadjuvantes tendo ações reativas). Para evitar *livelocks* os personagens podem ter objetivos que não dependam diretamente dos demais, ou caso dependam, que não sejam estados que representem ciclos contraditórios (casos como um personagem ter o objetivo de fugir e um outro ter o objetivo de alcançá-lo, sem um controle sobre o cansaço dos personagens). É possível também simular uma ação que duraria mais de um ciclo através da sua derivação em sequências de ações, cujas precondições e efeitos estejam logicamente interligados, no entanto essa abordagem ainda não seria suficiente para simular um combate, pois cada ação derivada deveria ser parte de um novo plano, o que contrasta com a escolha de uma única ação durativa.

3.1.2 Representação do Mundo

Neste trabalho, o processo de geração simula um mundo virtual como um sistema de estados cujas ações usadas para ir do estado inicial ao estado objetivo compõem a história. O processo é realizado através da descrição e manipulação dos elementos que compõem este mundo, nomeadamente: um mapa, um conjunto de objetos e um conjunto de personagens.

O mapa diz respeito ao cenário físico onde a história se passa e é composto basicamente por um grupo de lugares interligados por meio de passagens, sendo representado como um grafo não direcionado. Estes lugares abrigam os objetos e personagens que compõem o mundo e são descritos através de características próprias preestabelecidas que sejam relevantes para a história a ser gerada. As passagens, por sua vez, são representadas através de relações entre dois lugares (como as arestas do grafo) e são utilizadas pelos personagens para que possam se deslocar pelo mapa. Como o sistema resultante não se destina à apresentação de histórias por meio de elementos tridimensionais, a descrição dos objetos e personagens não engloba seu posicionamento em um espaço virtual por meio de coordenadas, mas apenas por meio da definição de um lugar presente no mapa.

Os objetos não possuem características próprias, sendo representados apenas como características de lugares ou de personagens. No contexto de uma história eles são utilizados para a descrição de objetivos e ações que envolvam objetos.

Por fim, a descrição dos personagens é feita pela determinação de um conjunto de características próprias também preestabelecidas, um conjunto de operadores e um objetivo. O conjunto de características próprias é usado como meio de representar personalidade (necessária para guiar o processo de planejamento) e relações com os objetos, o conjunto de operadores diz respeito às ações que os personagens podem executar e o objetivo é usado para determinar o quê o personagem tentará realizar, representando o papel que ele desempenha na história. A Figura 3.2 mostra o diagrama de classes usado pelo sistema para representar o mundo.

O sistema se utiliza destas descrições como meio de simular a evolução do mundo virtual seguindo o ritmo determinado pela linha do tempo dividida em turnos. As características necessárias na descrição dos lugares e dos personagens, somadas aos operadores criados para os personagens são usadas como domínio de planejamento em PDDL em cada turno. Neste sistema, os personagens atuam como se possuíssem conhecimento completo sobre o mundo virtual e por isso o processo de geração cria um problema de planejamento em PDDL a cada turno formado pela descrição de todas as características de todos os

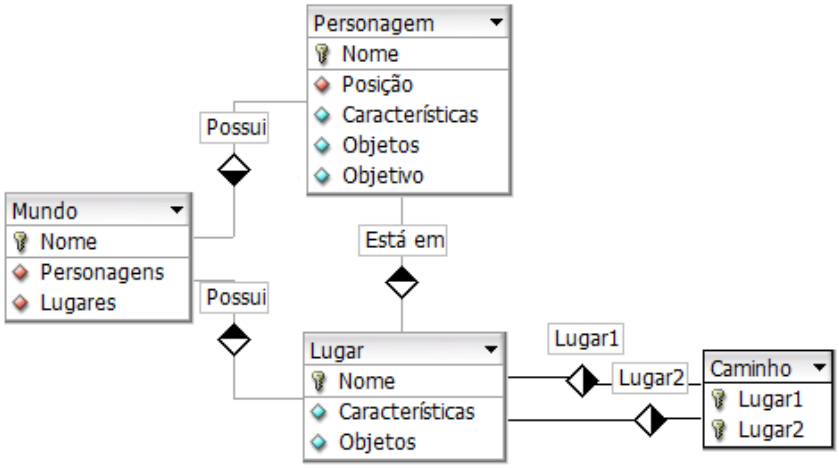

Figura 3.2: Diagrama de classes usado para a inserção de ruídos.

personagens e lugares, inserindo o objetivo do personagem dono do turno corrente como objetivo do problema. Além disso é utilizado um predicado específico(*actor*) cujo parâmetro recebe o nome do personagem corrente. Esse predicado é usado como precondição na descrição dos operadores para que o planejador se utilize apenas de ações do personagem atuante para resolver o problema em questão. O problema de planejamento gerado no turno, em conjunto com o domínio, são enviados ao planejador que retorna um plano cuja primeira ação é executada.

As características próprias dos lugares e personagens são mapeados em predicados binários ou funções numéricas de mesmo nome em PDDL. O posicionamento dos personagens e dos objetos, assim como as passagens são descritas através de predicados predefinidos. Por exemplo, o presente sistema faz a descrição mostrada na Figura 3.3 para representar, em um problema de planejamento, o mundo composto pelo mapa mostrado na Figura 3.4, no qual as posições de dois personagens são os pontos $forest1$ e $forest2$, as posições de dois objetos são os pontos $forest3$ e $forest4$, cada lugar possui um valor de "perigo" como característica e cada personagem possui um valor de "resistência" como característica. O predicado *actor* é usado para que as ações sejam executadas por apenas um personagem, indicando que se trata do seu turno.

O processo de planejamento seleciona uma sequência de ações suficiente para levar o personagem do estado em que se encontra ao estado objetivo, no entanto a este personagem é permitido realizar apenas a primeira ação do plano. Como cada personagem em seu próprio turno deve realizar uma ação (isso se o planejador for capaz de encontrar um plano que o leve ao objetivo), ao final de cada turno o mundo passa por modificações, portanto as ações de um plano gerado por um personagem em um dado turno podem não

```
(: objects character1 character2 - character house forest1 forest2 ... - location)
        object1 object2 - object)
    (: init
        (adjacent house forest1)
        (adjacent house forest2)
        (adjacent forest1 forest2)
        ...
        (actor character1)
        (at character1 forest1)
        (at character2 forest2)
        (in object1 forest3)
        (in object2 forest4)
        (= (stamina character1) 2)
        (= (stamina character2) 6)
        (= (danger house) 0)
        (= (danger forest1) 3)
        ...
    )
    (goal (and (...)))
)
```

Figura 3.3: Representação de um estado do mundo como um problema PDDL.

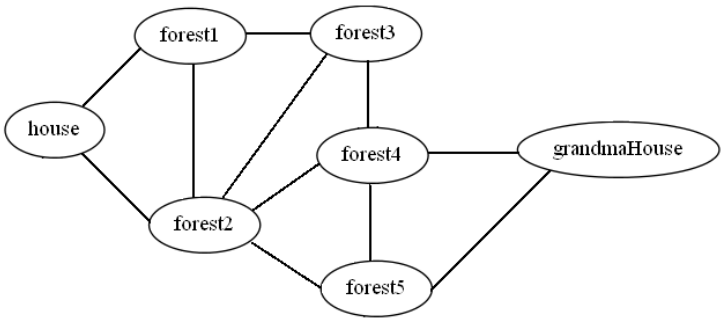

Figura 3.4: Exemplo de mapa

ser mais executáveis no seu turno seguinte, o que torna necessária a execução do processo de planejamento a cada turno.

Cada personagem possui seu próprio objetivo que pode vir a interferir nos objetivos dos demais, tal como matar outro personagem, ou pode se referir apenas a si próprio, tal como pegar um determinado item. Não sendo capazes de obter novos objetivos, a chegada a um estado objetivo que esteja em conflito com os demais pode inviabilizar a ação dos demais personagens e, como consequência, determinar o fim da história. Se não

houver objetivos conflitantes, o sistema pode continuar a executar até que todos sejam alcançados.

A evolução das pesquisas dentro da área de planejamento tem resultado em algoritmos planejadores com performance suficiente para aplicação, mesmo em sistemas de storytelling que requisitam a geração de planos em tempo de interação. No entanto, o foco deste trabalho inicial não envolve um processo interativo, mas sim a análise de histórias que possam ser geradas pelo uso de ações incorretas, o que requer um compromisso com a geração de planos ótimos que possam ser usados na comparação entre um plano ótimo gerado com informações incorretas, ou incompletas, e um plano ótimo gerado com informações íntegras. Foi escolhida a implementação de Zeyn Saigol [26] do algoritmo de *GraphPlan* como planejador para este sistema, devido à garantia de otimalidade (o que neste caso, onde considera-se que todas as ações têm o mesmo custo, se refere unicamente à quantidade de ações em um plano) e a disponibilização do código de forma estruturada, o que contribuiu para ser usado no código deste sistema.

3.2 Ruído

A abordagem baseada em personagens trabalha sem a utilização de enredo pré-estabelecido, baseando-se apenas nas descrições dos elementos que compõem o mundo, com foco em seus personagens. Estas descrições devem ser suficientemente detalhadas de forma a permitir que um personagem seja capaz de tomar decisões, expressar sua personalidade e gerar histórias diferentes. O aumento da quantidade de características e ações usadas para descrever um personagem pode tornar sua atuação mais variada, mas não necessariamente mais realista. Este fato justifica a pesquisa por elementos diversificados, como é o caso do ruído, que enriqueçam os comportamentos dos personagens.

A presente pesquisa considera como incorreta qualquer ação que não seria escolhida por um personagem se ele fosse onisciente. Para fazer com estas ações sejam parte dos planos dos personagens, visto que eles têm conhecimento pleno do mundo, uma vez por turno o sistema realiza uma modificação não cumulativa na descrição do mundo usada como conhecimento do personagem atuante removendo informações ou trocando seus valores. A essas variações foi dado o nome de ruído, sendo o primeiro caso chamado de ruído por omissão e o segundo de ruído por divergência. Na Figura 3.5 é apresentado o diagrama que demonstra os processos executados pelo sistema no decorrer de um turno com a inserção de ruídos: a descrição corrente do mundo passa por uma modificação

causada pela aplicação de ruídos, e em seguida a descrição modificada é usada em conjunto com o objetivo do personagem atuante para compor um problema de planejamento. O problema é enviado junto ao domínio para o planejador, que retorna um plano, cuja primeira ação é executada causando uma modificação no estado do mundo.

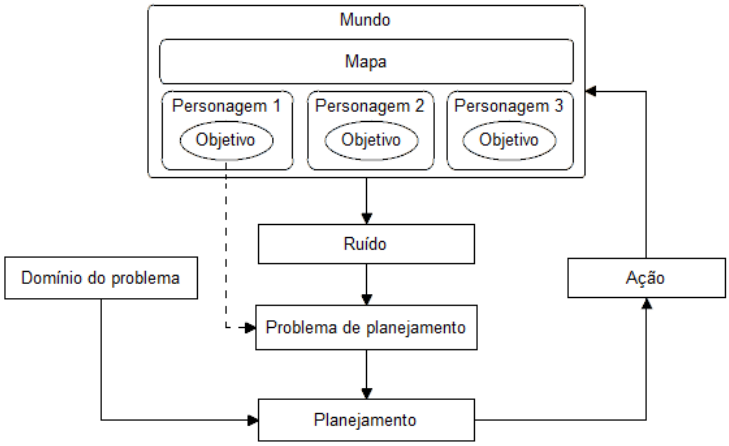

Figura 3.5: Diagrama de processos de um turno com inserção de ruídos

3.2.1 Ruído por Omissão

Os ruídos por omissão são usados para que os personagens utilizem planos alternativos ao ótimo para alcançarem seus objetivos e, diferente de simples remoções, eles são controlados de forma a manter informações suficientes para que não inviabilizem estes objetivos. Considerando, por exemplo, um mundo composto por um personagem e uma série de salas interconectadas que podem ou não conter determinados itens, o ruído por omissão poderia ser usado para remover a informação da posição de alguns deles. Se o personagem tiver como objetivo a coleta de uma quantidade qualquer de itens, ele pode se ver obrigado a atravessar um número maior de salas simplesmente por ter perdido a posição daqueles que estavam mais próximos.

A determinação de quais informações podem ser removidas por si só pode ser uma tarefa desafiadora dependendo da complexidade da descrição do mundo e dos objetivos. No exemplo citado acima, é necessário garantir que o personagem conheça a localização de uma quantidade de itens maior ou igual ao determinado pelo seu objetivo e conheça o mapa o suficiente para percorrê-lo em busca dos itens. A escolha de quais informações podem ser retiradas é realizada pelo sistema randomicamente, porém é manipulada de forma a manter os nós do grafo que representam o mapa com o grau mínimo necessário para mantê-lo conexo e, especificamente para o caso dos testes realizados, a não omitir as informações envolvidas em seus objetivos.

O uso do ruído por omissão parece, à primeira vista, causar necessariamente um aumento no tamanho da história original pela retirada de informações que permitiriam a um personagem atingir o seu objetivo de forma mais direta. No entanto, em casos onde os personagens têm objetivos conflitantes, uma dificuldade gerada para um personagem pode ajudar os demais a cumprirem seus próprios objetivos e consequentemente a finalizar a história prematuramente.

3.2.2 Ruído por Divergência

Enquanto o erro por omissão pode levar um personagem a ter uma quantidade menor de opções de ações, ou principalmente a executar ações que mesmo não sendo corretas ainda estejam de acordo com seus objetivos, o ruído por divergência tem como motivação a execução de ações que o afastem destes objetivos. Já que os personagens são criados de forma a serem capazes de responder a cada situação em que possam se encontrar com um grupo de ações formados por precondições bem definidas, se as informações que eles possuem para selecionar suas ações estiverem erradas é possível que eles selecionem ações que venham a dificultar o cumprimento dos seus objetivos.

Voltando ao exemplo descrito para exemplificar o ruído por omissão, uma possibilidade de utilização do ruído por divergência seria a alteração do conhecimento que o personagem tem à respeito das passagens do mapa pela mudança de quais lugares estão ligados entre si, ou ainda a alteração do que se sabe sobre as posições de personagens e objetos. Isso faria o personagem visitar inutilmente um conjunto de salas que ele acreditaria possuir os itens que está buscando ou ainda se perder no mapa através do uso de passagens com informações incorretas. Diferente do erro gerado pela retirada de passagens que ainda permite ao personagem criar planos aplicáveis, o erro gerado pela troca de passagens faz com que o personagem use caminhos que levem a salas não previstas ou mesmo que não existam. O ruído por omissão pode ser considerado mesmo um caso particular do ruído por divergência, no qual a troca de uma informação se dá pela sua negação. Neste trabalho eles foram tratados de forma diferenciada para explicitar o controle necessário em ambos os casos e a adaptação deles dentro da simulação de percepções tratada no Capítulo 4.

O uso do ruído por divergência nas passagens do mapa requer a administração das escolhas feitas pelo personagem sobre o seu ponto de vista e de suas consequências reais, o que se encontra fora do escopo deste trabalho inicial. Dessa forma, o erro por divergência é aplicado apenas ao conhecimento que os personagens têm a respeito da posição dos objetos do mundo.

3.3 Testes

3.3.1 Cenário

Foi escolhido como caso de teste a geração das ações envolvendo um segmento da história Chapeuzinho Vermelho. O ato é composto pela atuação de dois personagens: a protagonista e o Lobo. A história se passa em uma floresta a qual Chapeuzinho deve atravessar para chegar a casa de sua avó.

Para este cenário, o mapa mostrado na Figura 3.4 foi criado para representar a floresta, na qual estão espalhados alguns campos de flores. Chapeuzinho inicia a história em sua própria casa (*house*) com o objetivo de chegar à casa da avó (*grandmaHouse*) com uma certa quantidade de flores. O lobo inicia a história em um ponto no interior da floresta ($forest5$) escolhido para representar um ponto qualquer do mapa que não lhe dê muita vantagem, com o objetivo de matar a protagonista. Como ela não será capaz de se deslocar depois de ser morta, os objetivos são conflitantes e o sistema cessa a criação da história assim que um deles for alcançado.

As ações disponibilizadas a ambos os personagens são mover-se entre localizações adjacentes, matar um personagem que esteja na mesma localização e pegar objetos que estejam na mesma localização. A Figura 3.6 detalha o domínio usado para a geração da história com estas configurações.

Com esta descrição é possível gerar diferentes sequências de ações com base nos caminhos que os personagens escolherem atravessar. Chapeuzinho tem o problema extra de obter algumas flores antes de chegar à casa da avó, o que dá ao lobo mais oportunidades de concretizar seu objetivo. Para habilitar novas possibilidades de desenvolvimento, o erro por omissão é utilizado nos conhecimentos que os personagens têm do mapa para remover informações relacionadas às passagens. São removidas, por turno, duas ou três passagens que não estejam ligadas a um mesmo nó. Isso leva a uma redução na possibilidade de movimentação. Porém, considerando que a cada turno diferentes informações sobre passagens são removidas, um caminho que havia sido pensado por um personagem pode passar a ter becos sem saída, que resulta em histórias diferentes pela necessidade de novos planos corretivos.

O ruído por divergência é utilizado para modificar a posição dos campos de flores, o que leva a protagonista a perder mais ou menos tempo em sua busca. O controle feito sobre o ruído de divergência é usado para manter informações a respeito do posicionamento das

```
(define (domain story)
    (: types location character object)
    (: predicates
        (near ?l1 ?l2 - location)
        (in ?o - object ?l - location)
        (at ?c - character ?l - location)
        (hold ?c - character ?o - object)
        (dead ?c - character)
        (actor ?c - actor)
    )

    (: action move
        : parameters (?c - character ?from ?to - location)
        : precondition (and (actor ?c) (not (dead ?c)) (near ?from ?to) (at ?c ?from))
        : effect (and (not (at ?c ?from)) (at ?c ?to))
    )
    (: action kill
        : parameters (?c1 ?c2 - character ?l - location)
        : precondition (and (actor ?c1) (not (dead ?c1)) (at ?c1 ?l) (at ?c2 ?l))
        : effect (and (dead ?c2))
    )
    (: action pick
        : parameters (?c - character ?l - location ?o - object)
        : precondition (and (actor ?c) (not (dead ?c1)) (at ?c ?l) (in ?o ?l))
        : effect (and (not (in ?o ?l)) (hold ?c ?o))
    )
)
```

Figura 3.6: Domínio usado para a descrição do mundo.

flores suficiente para que Chapeuzinho possa pegar a quantidade que for necessária.

Para avaliar a utilidade desses dois tipos de ruídos, o processo de geração de histórias foi realizado em 4 situações diferentes: sem o uso de ruídos, com a inserção apenas do ruído por omissão, com a inserção apenas do ruído por divergência e por fim com ambos os ruídos. Para os testes que se utilizaram de algum ruído foram realizadas 30 execuções, cujos resultados apresentaram a variabilidade esperada.

3.3.2 Resultados

Como um algoritmo determinístico, o GraphPlan é capaz de gerar apenas uma história para o caso de teste que não se utiliza de ruídos. Ela é composta de 6 ações, com as quais o Lobo foi capaz de terminar vitorioso. A sequência de ações resultante é mostrada a seguir:

[**Chapeuzinho vai para floresta** 2, **Lobo vai para floresta** 2,
Chapeuzinho vai para floresta 5, **Lobo vai para floresta** 5,
Chapeuzinho pega uma flor, **Lobo mata Chapeuzinho**]

Com o ruído por omissão foi possível obter 10 variações, dentre os quais 4 terminaram com a morte da protagonista e 6 na sua chegada à casa da avó. Das 30 execuções, 24 resultaram nos finais onde o Lobo atinge seu objetivo e 6 nos finais onde Chapeuzinho chega na casa de sua avó. Em específico, foi gerada uma história com um número reduzido de ações onde Chapeuzinho moveu-se diretamente para a posição do Lobo, o que comprova a capacidade de se gerar histórias com tamanhos variados. A seguir é explicitada uma das sequências geradas a partir deste ruído, na qual mesmo o Lobo tendo alcançado Chapeuzinho logo no início (primeira linha), não pôde acompanhá-la por perder a informação de uma das passagens que ela utilizou (segunda linha) e, por isso não pôde atacar.

[**Chapeuzinho vai para floresta** 2, **Lobo vai para floresta** 2,
Chapeuzinho vai para floresta 4, **Lobo vai para floresta** 5,
Chapeuzinho pega uma flor, **Lobo vai para floresta** 4,
Chapeuzinho vai para Casa da Avó]

Para o segundo caso, no qual foi utilizado apenas o ruído por divergência, o processo gerativo criou 8 desenvolvimentos diferentes, tendo 5 deles terminado com a vitória do Lobo e 3 com a vitória da protagonista. Das 30 execuções, 24 apresentaram os finais favoráveis ao lobo e 6 os finais favoráveis a Chapeuzinho. A história mostrada a seguir tem final favorável ao Lobo, mas diferente do final obtido sem o uso de ruídos, o que é resultado da troca da posição do campo de flores:

[**Chapeuzinho vai para floresta** 2, **Lobo vai para floresta** 2,
Chapeuzinho vai para floresta 5, **Lobo vai para floresta** 5,
Chapeuzinho vai para floresta 2, **Lobo vai para floresta** 2,
Chapeuzinho pega uma flor, **Lobo mata Chapeuzinho**]

No caso final, com os dois tipos de ruídos, o sistema foi capaz de encontrar 15 desenvolvimentos distintos, tendo 6 deles terminado com a morte da protagonista e 9 com

o alcance do seu objetivo. Das 30 execuções, 18 apresentaram os 6 primeiros finais e 12 apresentaram os outros 9 finais. Este caso foi capaz de encontrar a seguinte história alternativa com uma quantidade consideravelmente maior de ações.

[**Chapeuzinho vai para floresta** 2, **Lobo vai para floresta** 5,
Chapeuzinho vai para floresta 3, **Lobo vai para floresta** 2,
Chapeuzinho vai para floresta 4, **Lobo vai para floresta** 4,
Chapeuzinho vai para floresta 3, **Lobo vai para floresta** 2,
Chapeuzinho vai para floresta 4, **Lobo vai para floresta** 4,
Chapeuzinho vai para floresta 5, **Lobo vai para floresta** 5,
Chapeuzinho pega uma flor, **Lobo mata Chapeuzinho**]

A quantidade média de ações das histórias criadas com o uso exclusivo do ruído de omissão foi de 6,40, no caso do ruído de divergência foi de 6,38 e 7,00 com o uso de ambos. Apesar das médias se encontrarem próximas ao tamanho das histórias geradas no caso sem ruídos, elas são resultado de uma distribuição equivalente entre histórias com um número maior e com um número menor de ações. Os testes foram executados em um computador equipado com um Core 2 Duo E8400 de 3.0GHz e 2GB de memória RAM DDR2. O tempo gasto para o caso sem ruídos foi de aproximadamente 14 segundos. O tempo médio da geração de histórias para o caso com o ruído por omissão foi de 12 segundos, para o caso com o ruído por divergência foi de 51 segundos, e para o caso com ambos os ruídos foi de 234 segundos, porém uma das histórias geradas neste último caso precisou de 97 minutos para ser finalizada.

3.4 Discussão

O Lobo inicia a história com a vantagem de se encontrar entre a posição inicial da protagonista e a casa da avó, por isso a maioria das histórias, incluindo a gerada no caso sem ruídos, terminou a seu favor. O erro por omissão foi capaz ajudar Chapeuzinho, fazendo com que os personagens se esquecessem de algumas passagens que os levariam a um encontro prematuro, o que deu à protagonista uma distância segura para pegar as flores e seguir com seu objetivo. No entanto, em uma das histórias o resultado foi o inverso: a protagonista seguiu diretamente para a localização do lobo por perder algumas de suas opções de movimento.

O uso apenas do erro por divergência não foi capaz de prevenir a perseguição do lobo, mas em alguns casos permitiu que Chapeuzinho coletasse flores em posições mais

próximas a sua posição inicial, o que reduziu as chances do Lobo alcançá-la. Em outras situações, as posições das flores eram omitidas assim que a personagem tentava coletá-las, o que forçava a continuação da busca e consequentemente o aumento das histórias. Por fim, com ambos os ruídos, diferentes situações emergiram, como casos onde a posição das flores era trocada para lugares cujos caminhos eram esquecidos, o que em condições diferentes contribuiu com os objetivos de ambos os personagens.

Pelas variações causadas ao desenvolvimento de uma história, a modelagem de ruídos se mostrou útil no contexto de storytelling. Dentro do cenário utilizado, com uma quantidade limitada de operadores, o uso de informações erradas resultou basicamente na modificação da trajetória dos personagens, mas que no caso era o elemento principal para a definição da conclusão da história. Em cenários mais complexos o ruído poderia resultar em atuações mais diversificadas com um número maior de troca de ações. Para que possa ser aplicado adequadamente ao contexto de storytelling, é necessário que o ruído seja incorporado à história através de um mecanismo de controle automatizado que justifique a sua inserção e que possa gerá-lo de forma não aleatória e coerente com os personagens.

Capítulo 4

Simulação de Percepções

O capítulo anterior apresentou uma forma controlada de inserção de erros nas bases de conhecimento dos personagens, de forma a causar o planejamento e a execução de ações incorretas que resultam em variabilidade no processo de geração de histórias. A inserção de erros neste caso, não é justificada no contexto da história e foi implementada para analisar as consequências e necessidades de sua utilização. Este capítulo descreve o método desenvolvido nesta pesquisa com o intuito de possibilitar a execução de ações incorretas no processo de geração de histórias baseado em personagens e, como extensão ao trabalho anterior, é inserida a simulação de percepção como método de geração de erros dinamicamente.

As seções seguintes descrevem as extensões adicionadas à arquitetura do processo de geração de histórias (seção 4.1), o que envolve um modelo de descrição de mundo que viabilize o uso de percepções (seção 4.1.1) e a utilização de regras de comportamento e de eventos (seção 4.1.2), os processos responsáveis pela simulação de percepções em conjunto com o tratamento de ações incorretas (seção 4.2) e, por fim, o teste desenvolvido para validar o método descrito seguido da análise dos resultados obtidos (seção 4.3).

4.1 Arquitetura

A nova arquitetura mantém a divisão da linha do tempo em turnos, visto que ela não inviabiliza o processo perceptivo e facilita a sua aplicação. Caso fossem utilizadas ações com duração explícita, os personagens teriam que recorrer ao processo perceptivo continuamente e isso afetaria não só o desempenho geral do sistema como o tempo de resposta de cada um deles.

Também é mantida a linguagem PDDL como meio de representação para que o sistema possa se utilizar de planners existentes. No entanto, como um dos próximos passos ao sistema de geração desenvolvido nesta pesquisa seria integrá-lo ao contexto de storytelling interativo, é necessário verificar a influência do processo perceptivo no desempenho do sistema. Para este caso foi escolhida a 6ª versão do SGPlan (última versão disponibilizada anterior a execução dos testes) como algoritmo de planejamento devido à eficiência que ele tem demonstrado que permitiria sua utilização mesmo em sistemas de storytelling interativo [22].

As principais mudanças na arquitetura dizem respeito a representação do mundo, a adição de regras de comportamento e de execução de eventos, e a inserção do processo perceptivo.

4.1.1 Representação do mundo

A representação de um mundo virtual nesta extensão também se divide na descrição de um mapa, de um conjunto de objetos e de um conjunto de personagens. O mapa segue a descrição do trabalho anterior sendo divido em lugares interligados, porém sem a utilização de características próprias. A descrição é feita como um conjunto de lugares que são formados pelos seguintes elementos:

- ID: valor exclusivo usado para identificar o lugar, criado pelo sistema no início da execução.
- Nome: usado para referenciar o lugar em questão.
- Vizinhos: lista de lugares que estão conectados diretamente a este, isto é, sem a necessidade de interação com um objeto, como por exemplo uma porta.
- Passagens: lista de lugares que estão conectados indiretamente a este, isto é, que exijam a interação com um objeto.

Os personagens representam a base do processo de geração e, por isso, têm uma estrutura mais detalhada, feita através dos elementos:

- ID: valor exclusivo usado para identificar o personagem, criado pelo sistema no início da execução.
- Nome: usado para referenciar o personagem em questão.

- Atributos: descrição das propriedades e relações inerentes ao personagem. Dizem respeito a qualquer característica ou condição que pode ser usada no decorrer da história por ações e/ou eventos.
- Localização: nome do lugar onde o personagem se encontra.
- Ações: operadores de planejamento que representam as ações que o personagem sabe executar.
- Objetivos: lista de objetivos ordenada de acordo com a prioridade.
- Percepção: valor numérico que determina sua qualidade de percepção (seu uso é descrito em detalhes na seção 4.2).
- Componentes: descritores da composição física do personagem (seu uso é descrito em detalhes na seção 4.2).
- Modelos de personagens e objetos: padrões conhecidos de personagens e objetos usados para a determinação dos atributos de elementos desconhecidos (seu uso é descrito em detalhes na seção 4.2)
- Conhecimento do mundo: objetos, personagens e lugares conhecidos. Determina os valores dos atributos e dos componentes que se acredita ser verdade sobre os demais personagens e objetos, além do que se sabe sobre o mapa. Pode-se interpretar como uma representação do mundo sob o ponto de vista do personagem.

Os objetos representam literalmente os itens dispostos no mapa, sendo formados por uma estrutura menor composta também por um ID, um nome, um conjunto de atributos, um conjunto de componentes e uma localização. Além destes elementos é necessária a definição do tipo do objeto. Nesta nova representação, os objetos são organizados em tipos para que seja possível descrever ações e atributos que façam referência a objetos de tipos específicos ou mesmo a todos os objetos.

A Figura 4.1 apresenta o novo diagrama de classes, explicitando a diferença entre a descrição do mundo e do conhecimento dos personagens. Nesta figura os atributos e componentes são apresentados apenas como variáveis dos personagens e objetos, sendo no entanto implementadas como classes separadas.

Como uma das extensões, foi inserida a possibilidade de especificação do mapa, dos objetos e dos personagens em estruturas textuais próprias, desenvolvidas para que seja possível definir o mundo virtual para o qual se deseja criar uma história. A seguir,

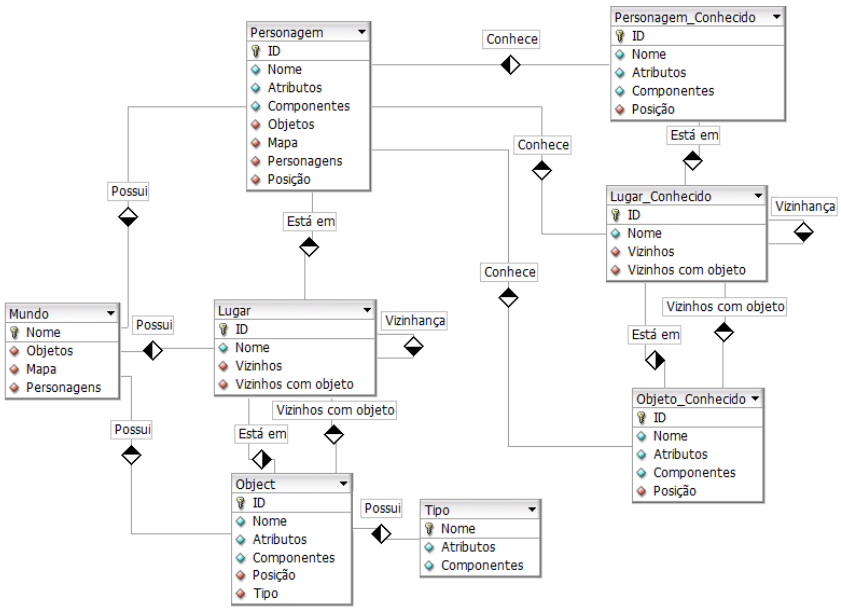

Figura 4.1: Diagrama de classes usado para a simulação de percepção

é apresentado como estas estruturas podem ser usadas para criar o estado inicial do mundo, de forma a seguir as especificações descritas acima e como elas estão ligadas à representação de problemas em PDDL usada pelo sistema.

4.1.1.1 A criação do mundo virtual

Os atributos que formam os objetos e personagens são particularmente importantes para a simulação de percepções. Eles são definidos para comportar tipos diferentes de valores com base na representatividade da PDDL3.

Usando-se de PDDL é possível representar um mundo como sistema de estados por meio de predicados e funções que definem as suas propriedades e relações. Predicados e funções são criados através da definição de um nome que sirva como referência e da definição de um grupo de parâmetros ao qual eles possam ser aplicados. Um predicado definido através de um grupo formado por apenas um parâmetro é denominado unário e usado para estabelecer uma propriedade sobre este parâmetro. Um predicado definido com um grupo formado por n parâmetros é denominado n-ário e usado para estabelecer uma relação entre os parâmetros. Já as funções são usadas para mapear o conjunto de parâmetros a um valor ou a um outro parâmetro, o que também pode ser interpretado como o estabelecimento de uma propriedade ou de uma relação.

A definição de um atributo para criação de objetos e personagens é feita através da definição de um nome e da escolha de um tipo dentre os predefinidos pelo sistema. São dis-

ponibilizados quatro tipos de atributos, nomeadamente: booleano, numérico, lista e lista numérica. O tipo booleano é usado para representar propriedades cujo valor possa ser expressado como *verdadeiro* ou $falso$; o tipo numérico é usado para representar propriedades cujo valor possa ser expressado através de um número inteiro; o tipo lista é usado para representar a existência de uma relação entre o personagem ou objeto em questão e um grupo de elementos; e, por fim, o tipo lista numérica é usado de forma semelhante ao tipo lista, porém definindo um valor numérico para a relação representada. Estes dois últimos tipos exigem a determinação da espécie dos elementos com o qual a relação é formada. Como exemplo de utilização destes tipos na criação de atributos, a sequência abaixo ilustra 4 atributos que poderiam ser usados na criação de um personagem.

```
boolean dead false;
numeric stamina 5;
list:object hold basketofsweets flower;
intlist:character friendship Grandma:20;
```

Esta descrição poderia ser usada para especificar um personagem que não está morto, possui valor de resistência 5, está de posse de uma cesta de doces e de um punhado de flores, e tem uma relação de amizade de valor 20 com o personagem Grandma. A escolha pela utilização destes quatro tipos como formas de determinar os valores dos atributos foi feita tendo em vista o mapeamento desta descrição na criação de domínios e problemas de planejamento.

Nas Figuras 4.2 e 4.3 são apresentadas parte da definição de um domínio e de um problema que seriam resultado do mapeamento da lista de atributos apresentada acima como parte de um personagem. O tipo booleano, usado para criar o atributo *dead*, é mapeado em um predicado unário que recebe um personagem como parâmetro; o tipo lista, usado para criar o atributo *hold* é mapeado em um predicado n-ário que recebe como parâmetros um personagem e aquilo que for determinado pela lista. A instanciação no problema é feita para cada valor dado ao atributo, neste caso para os dois objetos $basketofsweets$ e $flower$. Na situação mencionada, o atributo foi criado para estabelecer um relação entre o personagem e um objeto, mas poderia ser criado para estabelecer uma relação entre o personagem, um objeto e um lugar por exemplo, e o atributo seria mapeado em um predicado com três parâmetros ao invés de dois. Os tipos numérico e lista numérica, usados para criar os atributos *stamina* e $friendship$ seguem os tipos booleano e lista, respectivamente, porém por meio de funções ao invés de predicados.

Os componentes, por outro lado, representam as características físicas de um elemento e não são usados pelo sistema na etapa de planejamento, mas sim no processo de percep-

```
(define (domain story)
    (:types location character object)
    (:predicates
        (dead ?c - character)
        (hold ?c - character ?o - object)
    (:functions
        (stamina ?c - character)
        (friendship ?c - character ?c - character)
    )
)
```

Figura 4.2: Domínio criado automaticamente englobando os atributos do personagem.

```
    (:objects nomedopersonagem - character)
    (:init
        (not (dead nomedopersonagem))
        (hold nomedopersonagem basketofsweets)
        (hold nomedopersonagem flower)
        (= (stamina nomedopersonagem) 5)
        (= (friendship nomedopersonagem Grandma) 20)
```

Figura 4.3: Problema criado automaticamente englobando os atributos do personagem.

ção, que é realizado à parte, como base para que um personagem tente entender os valores dos atributos dos demais elementos do mundo, o que será explicado mais adiante neste capítulo. Cada personagem e objeto deve ter em sua descrição um ou mais componentes, como apresentado na sequência abaixo, sendo cada um criado através da definição de um nome, um valor em *string* que o caracterize e um valor numérico que represente seu apelo visual. Os componentes são divididos em dinâmicos e estáticos para representar características que deveriam ou não mudar no decorrer da história, respectivamente. Dessa forma a criação de um componente também deve explicitar a qual grupo ele pertence. Nos exemplos abaixo, *Face* e *Cloth* representam os nomes dos componentes, *LittleGirl* e *RedHidingHood* os valores que os caracterizam, 6 e 9 representam seus valores de apelo e as indicações *false* e *true* explicitam se os componentes são dinâmicos ou não.

```
Components:
dynamic: false
Name: Face
Value: LittleGirl
Appeal: 6

dynamic: true
Name: Cloth
Value: RedHidingHood
Appeal: 9
```

Dadas as descrições de atributos e componentes, para a especificação dos objetos é necessário, em primeiro lugar, a enumeração dos tipos de objetos através dos atributos e componentes que lhes são característicos. Com os tipos definidos, um objeto é criado pela determinação do seu tipo e dos valores de seus atributos e componentes. Objetos também são caracterizados como estáticos ou dinâmicos, o que é usado para explicitar a possibilidade de um objeto alterar sua localização no decorrer de uma história. Se objetos de um determinado tipo não podem sofrer mudanças de localização, então eles devem ser descritos como estáticos, caso contrário como dinâmicos. As sequências seguintes mostram um exemplo de como definir um tipo e um objeto, respectivamente:

```
Type: door
dynamic: false

Attributes:
boolean locked
numeric lock

Components:
dynamic: false
Name: Body

dynamic: true
Name: Lock
```

```
Object doora1
type: door

Attributes:
locked false
lock 14

Components:
Body
Value: Dungeon
Appeal: 5

Lock
Value: true
Appeal: 2
```

Para a definição de um mapa, é necessária a enumeração dos lugares que o compõe, na qual devem ser detalhados, para cada lugar, os objetos que podem ser encontrados, seus vizinhos e passagens, quando existirem. O mapeamento do conjunto de lugares para PDDL se utiliza de predicados binários predefinidos para relacionar um personagem a um lugar, para relacionar um objeto a um lugar, para relacionar dois lugares que não estejam ligados por um objeto, e um predicado ternário predefinido para relacionar dois lugares que estejam ligados por um objeto.

A sequência abaixo exemplifica a definição de um mapa composto por três lugares, sendo dois deles ligados por uma porta e dois deles ligados diretamente. A Figura 4.4 mostra sua representação como um problema em PDDL, na qual cada lugar é instanciado como um objeto do tipo predefinido *location*, o predicado *near* é usado para relacionar dois lugares que estejam ligados diretamente, o predicado ternário *next* é usado para relacionar dois lugares que estejam ligados indiretamente (por meio de um objeto) e o predicado *in* é usado para relacionar um lugar a um objeto indicando a posição deste.

A criação de personagens também exige uma descrição extra, semelhante à dos tipos

```
Map CastleBasement
Places: littleredhouse forest1 forest2

Place: littleredhouse
Objects: doora1
Neighbors:
forest1 doora1

Place: forest1
Objects: doora1
Neighbors:
littleredhouse doora1
forest2

Place: forest2
Objects:
Neighbors:
forest1
```

$$
\begin{array}{l}
(:\ objects\ \ littleredhouse\ \ forest1\ \ forest2\ -\ \ location) \\
(:\ init \\
\quad (in\ \ littleredhouse\ \ doora1) \\
\quad (next\ \ littleredhouse\ \ forest1\ \ doora1) \\
\quad (in\ \ forest1\ \ doora1) \\
\quad (next\ \ forest1\ \ littleredhouse\ \ doora1) \\
\quad (near\ \ forest1\ \ forest2) \\
\quad (near\ \ forest2\ \ forest1)
\end{array}
$$

Figura 4.4: Problema criado automaticamente com a especificação do mapa.

de objetos, onde devem ser determinados os atributos e componentes que forem relevantes. Com esta descrição, cada personagem é criado pela definição dos valores de seus atributos e componentes, além dos demais itens enumerados acima, na estrutura de um personagem. As ações devem ser descritas seguindo o padrão de definição de ações em PDDL, que determina para uma série de parâmetros, um conjunto de precondições a serem atendidas e os efeitos da aplicação da ação aos parâmetros. O conhecimento inicial do mundo deve ser criado através da enumeração de objetos, personagens e locais, o que pode ser interpretado como o *background* do personagem. O que for especificado como conhecimento inicial é copiado para a base de conhecimento do personagem.

A Figura 4.5 mostra um exemplo de especificação de um personagem (no caso um exemplo de Chapeuzinho) com os elementos descritos até então. Novos conceitos são apresentados nas seções seguintes onde a descrição do personagem é complementada.

É importante ressaltar que o sistema mantém para cada personagem o seu próprio conhecimento, o que pode ser encarado como uma cópia incompleta e modificada da

```
Character LittleRedRidingHood

Attributes:
dead false;
courage 5;
hold basketofsweets;
friendship Grandma:20;

Place: a1
KnowPlaces: a1
KnowObjects: basketofsweets
KnowCharacters: Grandma

Components:
Face
Value: LittleGirl
Appeal: 6

Cloth
Value: RedRidingHood
Appeal: 8

Actions:
(:action pick
       :parameters (?c - character ?l ?o - object)
       :precondition (and (actor ?c) (at ?l ?c) (in ?l ?o))
       :effect (and (not (in ?l ?o)) (hold ?c ?o))
)

(:action unlock
       :parameters (?c - character ?l ?d - door ?k - key)
       :precondition (and (actor ?c) (at ?l ?c) (in ?l ?d) (hold ?c ?k) (=
          (locktype ?k) (lock ?d)))
       :effect (and (not (locked ?d)))
)
...
```

Figura 4.5: Descrição parcial de um personagem

representação de mundo. Dessa forma, são mantidas tantas cópias de mundo quanto personagens, além da cópia com as descrições reais do mundo.

Nesta pesquisa foi desenvolvido um mapeamento para a linguagem PDDL dos elementos de um sistema de geração baseado em personagens. Alguns exemplos de trabalhos com foco na especificação de histórias em linguagens que possam ser usadas em planejamento são [21], no qual é proposto a formalização para a linguagem PDDL dos elementos de um sistema de geração baseado em enredo, e [6], que detalha o formalismo usado para especificação dos elementos usados no sistema Logtell em Prolog.

4.1.2 Eventos e Comportamentos

Um problema constante dos sistemas geradores de histórias é a escolha de sequências de ações e/ou eventos que não sejam apenas coerentes, mas que também possam causar a

noção de narrativa, como realizado em [16] através do *drama manager*. Coerência pode ser interpretada simplesmente como a ordenação lógica entre as ações e eventos que compõem a história, feita através de suas precondições e efeitos, ou como a seleção de sequências que estejam de acordo com o que se espera dos personagens, do mundo e do desenvolvimento da história.

Pela abordagem baseada em personagens, onde a orientação do processo gerativo se dá pelos seus próprios objetivos e conhecimentos, a história final não necessariamente apresenta uma estrutura coerente e semelhante à de narrativas, como mencionado anteriormente. Além do que, a busca por poucos objetivos, ou objetivos que estejam distantes da situação inicial de cada personagem pode dificultar ou mesmo tornar impossível o processo de geração. Uma técnica usada para tratar este problema, e que foi adotada no sistema deste trabalho, é o uso de regras de inferência de objetivos [7]. Esta técnica simula comportamento fazendo com que os personagens persigam novos objetivos quando suas regras são atendidas. Dessa forma, é possível determinar regras para objetivos que os personagens devam alcançar na introdução, que, quando cumpridos, estabeleçam as condições para a inferência dos objetivos do desenvolvimento, que por sua vez viabilizem os objetivos da conclusão.

Essas regras não devem estar logicamente interconectadas de forma rígida para que seja possível gerar histórias com um certo nível de variabilidade. A utilização de comportamentos é viabiliza pelo sistema através da criação dessas regras seguindo o padrão estabelecido na PDDL para a descrição de ações, porém, com a descrição do estado objetivo no lugar da descrição de efeitos. O código a seguir ilustra um exemplo de descrição de comportamento, com o qual um personagem evitaria ficar em um mesmo lugar onde houvesse um personagem de má índole:

```
(:behavior notNearEvil
       :parameters (?c1 ?c2 - character ?l - location)
       :precondition (and (actor ?c1) (at ?l ?c1) (at ?l ?c2) (evil ?c2))
       :goal (and (not (at ?l ?c1)))
)
```

Cada personagem deve ter seu próprio conjunto de comportamentos, cujos objetivos são adotados sempre que as regras são atendidas. Para trabalhar com estas descrições de comportamento, o sistema cria um domínio para cada personagem onde no lugar das ações são inseridos seus comportamentos. No início de cada turno, após a atualização da base de conhecimentos do personagem, o sistema utiliza a implementação do GraphPlan, descrito no capítulo anterior, enviando-lhe o domínio de comportamentos e um problema de planejamento com o estado atual do mundo sob a visão do personagem para obter

um conjunto contendo todas as possíveis combinações de instâncias de comportamento. O sistema então verifica quais destas instâncias têm suas precondições atendidas, sob o ponto de vista do personagem. Os objetivos cujas precondições são atendidas passam a fazer parte da lista de objetivos do personagem. Logo após essa operação, o sistema remove, dentre todos os objetivos, aqueles que já tenham sido alcançados.

A seleção de qual objetivo o personagem tentará alcançar é feito através de uma distribuição de prioridade que segue a ordem na qual os objetivos foram adotados, de tal forma que o objetivo mais recente possui maior prioridade. Com a inserção dos comportamentos, os personagens passam a seguir o modelo de agentes BDI (beliefs, desires and intentions) [18], o qual estabelece que um agente possui sua própria visão de mundo (beliefs) e de um grupo de objetivos (desires), ele se compromete a realizar um subgrupo viável (intentions). Neste trabalho, objetivos considerados inalcançáveis não são abandonados, ao invés disso, sempre que um personagem tiver apenas objetivos que não possa cumprir, o sistema lhe dará como novo objetivo chegar a um determinado lugar que ele conheça, mas ainda não tenha visitado (escolhido randomicamente dentre os que ainda não foram visitados). Caso isso não seja possível, o personagem não agirá no turno em questão e, se nenhum personagem agir dentro de um mesmo ciclo, o sistema finaliza sua execução. Os comportamentos são usados para descrever apenas estados que se deseja alcançar, não sendo possível determinar como objetivo a manutenção de um determinado estado ou ainda a passagem por determinados estados durante a execução de um plano.

Também foi inserida a possibilidade de execução de eventos no intuito de simular reações do ambiente a determinados estados do mundo. No sistema proposto é permitida a definição de eventos também seguindo o padrão estabelecido para a descrição de ações em PDDL. Em adição, é possível especificar efeitos extras como a adição de informações ao conhecimento de algum personagem que tenha participado ou não do evento e mudanças nos componentes de algum objeto ou personagem. O código abaixo ilustra um exemplo de especificação de evento, onde uma armadilha é acionada caso alguém se aproxime de um determinado tesouro. Como efeitos adicionais, o personagem alvo passa a saber da existência da armadilha e tem sua aparência alterada.

```
(:event activateTraps
       :parameters (?c - character ?l - location ?t1 - trap ?t2 - treasure ?d -
          door)
       :precondition (and (at ?l ?c) (in ?l ?t1) (armed ?t1) (nearObj ?c ?t2)
          (installed ?t1 ?d))
       :effect (and (not (armed ?t1)) (locked ?d))
)

addKnowledge ?c
```

```
object trap

changeComponent
character ?c
Face
Value: Skull
```

A verificação da viabilidade dos eventos é feita de forma semelhante ao realizado com os comportamentos: o sistema gera um domínio contendo os eventos descritos pelo autor como ações e um problema de planejamento contendo a descrição do mundo. Ambos são enviados à implementação do GraphPlan para se obter um conjunto de eventos aplicáveis. Esse processo é realizado ao final de cada turno após a ação do personagem e engloba não o seu conhecimento de mundo, mas sim a descrição real. Todos os eventos que forem considerados aplicáveis são executados e adicionados a história. Os personagens são capazes de perceber todas as alterações feitas por um evento nos elementos do ambiente que eles tenham visto e que estejam em sua mesma localização.

É importante ressaltar a questão dos *livelocks* que se mantém, mas pode ser melhor controlada com a utilização de eventos e comportamentos: as mudanças de objetivo e de mundo causadas por eventos também podem resultar em personagens que nunca alcançam suas metas, pois mudanças de mundo podem estar sempre afastando um personagem de seu objetivo. No entanto, por ser possível designar novos objetivos a personagens, o autor pode criar comportamentos ou eventos que prevejam os possíveis *livelocks* de uma história e evitem o seu acontecimento. Para o caso da perseguição, como presente no capítulo anterior, onde uma história se vê presa um personagem fugindo e outro perseguindo, seria possível designar eventos que atrapalhem um dos personagens em alguns pontos do mapa, designar um comportamento de desistência por parte do perseguidor ou ainda um comportamento de reação por parte do perseguido.

4.2 Trabalhando com Erros de Percepção

Para esta pesquisa, a percepção de um personagem é definida como a recepção de informações dos elementos que compõem o ambiente onde ele atua, seguida da interpretação destas informações com possíveis mudanças de valores e sua inserção na base de conhecimentos do personagem. Dessa forma, uma observação ou interpretação incorreta pode levar o personagem a ter um conhecimento incorreto do mundo e, consequentemente, a executar ações incorretas. Esta seção detalha como a descrição do mundo é usada nos processos de observação e interpretação de forma a resultar em erros coerentes e, em seguida, como o sistema trata a execução de ações incorretas.

4.2.1 Percepção

A estrutura proposta para o mundo divide o cenário em lugares interconectados por passagens. Em cada lugar podem estar posicionados conjuntos de personagens e de objetos, que contêm em sua descrição conjuntos de componentes (como apresentado na seção 4.1.1). O processo de observação se utiliza dos componentes como unidade básica de percepção do mundo. Ao invés dos personagens perceberem os elementos do ambiente como um todo, como seria o caso de entrar em uma sala e imediatamente ter consciência de todos os objetos que a compõem, o sistema opera entregando aos personagens apenas os componentes destes elementos. Cada componente é criado com um valor numérico, denominado "apelo" que determina o quanto aquele componente pode ser distinguido do restante do ambiente. Cada personagem é criado com um valor numérico denominado "percepção", que representa a qualidade de sua visão.

O processo perceptivo qualifica como similares elementos que possuam ao menos um componente estático com valor comum e nenhum estático com valor distinto, ou ainda que possuam ao menos um componente dinâmico com valor comum e nenhum componente estático ou dinâmico com valor distinto. Entre elementos similares, o processo perceptivo considera como nível de similaridade a quantidade de componentes de valor comum, com ênfase nos componentes estáticos, de tal forma que dois elementos com apenas um componente estático de valor comum possuem maior similaridade que dois elementos com um número qualquer de componentes dinâmicos de valor comum.

Ao início de cada turno, o personagem que irá atuar recebe os componentes de todos os elementos que estiverem na mesma sala, mas fica apenas com aqueles cujo apelo for suficiente para sua visão (a Equação 4.1 detalha as condições necessárias para que um componente seja visto, onde Per_{max} indica o maior valor de percepção, Per_{chr} o valor de percepção do personagem em questão e *rand* um valor aleatório). Isso é feito para simular a dificuldade de observar completamente um ambiente e a necessidade que os humanos têm de interpretar dados incompletos. Foi usado o intervalo [0, 10] para atribuição dos valores de percepção e apelo, e [-1, 1] para os valores randômicos. A equação possui um fator randômico para evitar que os personagens sempre percebam exatamente os mesmos componentes quando se depararem com os mesmos elementos. Dessa forma é possível gerar variabilidade de histórias com base em mudanças na qualidade da observação.

$$Apelo > (Per_{max} - Per_{chr}) + rand \tag{4.1}$$

Com o conjunto dos componentes vistos, o personagem passa para a fase de interpretação, na qual deve tentar identificar quais são os elementos que os originaram, em outras palavras, o que ele viu. Este passo é dividido em duas fases. Na primeira fase, o personagem compara cada um dos componentes vistos de cada elemento com os componentes dos elementos que ele já conhece. Se ele encontrar um ou mais elementos similares, ele assume que o elemento encontrado de maior similaridade é o que ele viu, atualiza o seu número ID com o ID do elemento que originou os componentes semelhantes, atualiza os componentes dinâmicos do elemento encontrado com os componentes dinâmicos do elemento visto e atualiza a localização deste elemento com a sua própria localização. Caso contrário, ele assume que viu algo novo e cria uma nova instância em sua base de conhecimento com os componentes vistos. Quando um novo elemento é criado, o sistema passa para a segunda fase, onde o personagem tenta descobrir quais são os valores dos atributos do novo elemento. Isso é feito comparando os componentes do novo elemento com os componentes de padrões conhecidos.

Os padrões, aqui chamados de modelos, também são criados e definidos pelo autor seguindo a mesma estrutura usada para a criação de objetos e personagens, mas apenas com a descrição de atributos e componentes, e não necessariamente com todos. Eles são usados para explicitar as expectativas que um personagem possui à respeito de elementos desconhecidos, tendo por base suas características físicas, o que representa suas experiências e preconceitos com relação aos demais elementos do mundo. Segue abaixo um exemplo de definição de um modelo para personagens:

```
Model ghost

Attributes:
evil true
dead true

Components:
Body
Value: Transparent
```

Quando um novo elemento é criado na base de conhecimentos de um personagem, seus componentes são comparados com os componentes de cada modelo conhecido do mesmo tipo do elemento. Se modelos similares são encontrados, os valores de seus atributos são usados para preencher os valores dos atributos do novo elemento, em ordem de similaridade. Os atributos que não forem preenchidos dessa forma recebem um valor padrão característico de seus tipos: atributos booleanos recebem valor $falso$, numéricos recebem valor 0, listas e listas numéricas permanecem vazias. Seguindo este exemplo, se um personagem vê outro que não conhecia, cujo componente “Body” tenha valor “Transparent”, ele

acreditará que os atributos “evil” e “dead” do personagem desconhecido têm valor “true”.

Modelos possuem um nome próprio, como é o caso de “ghost” no exemplo acima, mas não necessariamente determinam um nome para os elementos que sejam similares. Caso não seja possível definir um nome para um novo elemento a partir dos modelos similares, o sistema utiliza uma *string* qualquer (tal como “a” ou “b”) de uma lista de nomes incógnitos predefinida em cada personagem. Esse recurso é usado pelo personagem para representar a idéia de que ele não tem pistas a respeito do nome do elemento que está analisando, e usado pelo sistema para ter uma forma de referenciar o elemento em questão durante o mapeamento do conhecimento do personagem em problemas de planejamento. Esse recurso também é utilizado para representar lugares não conhecidos do personagem. No início de cada turno, o personagem atualiza seu conhecimento a respeito do lugar onde se encontra com o nome correto e com os lugares que se conectam a ele. Os vizinhos que forem inseridos são armazenados na base de conhecimento com um nome proveniente da lista de incógnitos para indicar ao sistema que o personagem ainda não visitou o lugar. Vizinhos que se conectem ao lugar corrente do personagem por meio de um objeto são inseridos apenas se o personagem tiver visto o objeto em questão.

Elementos cujos componentes não tenham sido vistos pelo personagens são ignorados, enquanto os demais são guardados em uma lista específica criada para representar o contexto atual do personagem. O personagem usa esta lista inserindo elementos quando os tiver reconhecido ou criado, e esvaziando quando muda sua própria localização no mapa. Ao final do processo de percepção, o personagem faz uma busca por todos os elementos da sua base de conhecimento que supostamente se encontram em sua localização atual. Aqueles que não estiverem em sua lista de contexto têm sua informação de localização removida da base de dados. Em suma, se um personagem percebe um objeto em um turno, mas não o percebe no turno seguinte sem ter se movido para outro lugar e sem ter visto algum evento ou personagem mover o objeto, ele assume que o objeto ainda está na localização atual. Isso é feito para explicitar o contexto corrente, e para que a utilização do fator randômico da observação não afete o personagem fazendo-o adicionar e remover a localização de um objeto repetidamente.

Objetos estáticos são tratados de forma excepcional, uma vez reconhecidos eles não são confundidos com outros objetos e nem se admite que sua localização possa ser alterada, no entanto seus atributos ainda podem ser interpretados de forma incorreta. Pode-se dizer que objetos estáticos são reconhecidos pela localização e não pela aparência, assim eles podem ter componentes estáticos trocados durante a história e ainda serem reconhecidos.

A observação e a interpretação usadas para reconhecimento seguem a premissa de que existem objetos iguais, e os personagens são únicos. Em outras palavras, no caso do reconhecimento de objetos, mesmo que os componentes relacionados a uma observação sejam iguais aos de um objeto conhecido, eles serão tratados como parte de um novo objeto caso sejam encontrados em um lugar diferente do objeto conhecido, enquanto que no caso dos personagens, a localização destes componentes não afeta o reconhecimento. Dessa forma, o mesmo objeto visto em lugares diferentes será interpretado como dois objetos distintos, enquanto que dois personagens diferentes com componentes iguais serão tratados como se fossem o mesmo, caso sejam vistos em locais distintos.

Esse processo de interpretação também é executado sempre que um evento altera os componentes de um elemento. Quando uma alteração é realizada, todos os personagens que estiverem na mesma localização do elemento e o tiverem visto em um turno anterior tentarão comparar seus novos componentes com modelos conhecidos. A Figura 4.6 mostra uma visão esquemática dos processos que envolvem a percepção e o Algoritmo 4.1 apresenta uma simplificação do algoritmo usado na percepção de objetos ao início de um turno. Ele se diferencia da percepção de personagens apenas por tratar o reconhecimento de um objeto estático mesmo após troca de componentes estáticos, e por analisar se o objeto serve como passagem entre localizações.

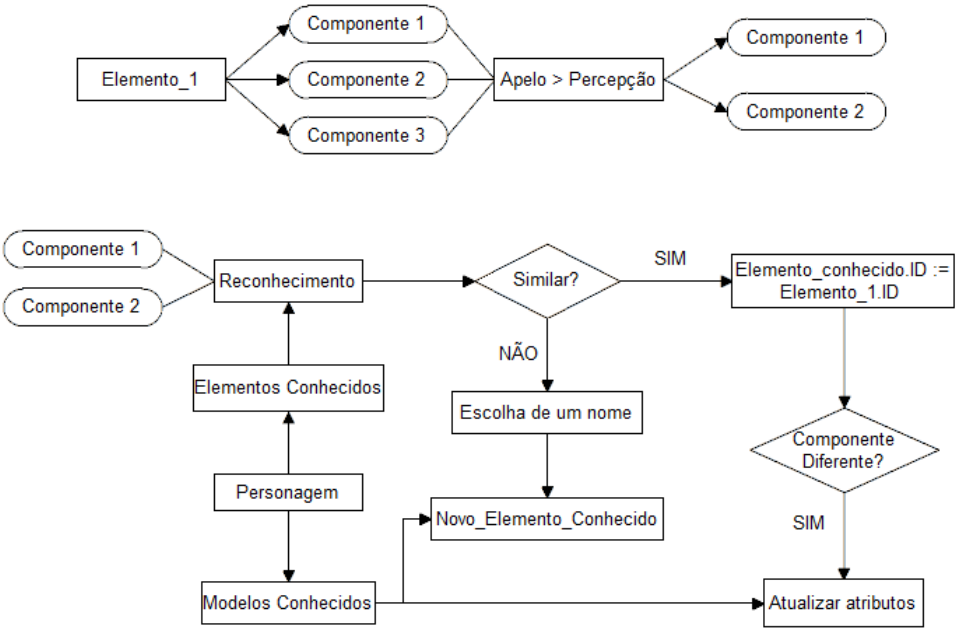

Figura 4.6: Diagrama dos processos de observação e interpretação, respectivamente

Os erros possibilitados pelo processo de percepção são resultado de uma observação incompleta do ambiente, o que permite ao personagem associar elementos a padrões

Algoritmo 4.1 Algoritmo de percepção de objetos.

Entrada: *mundo*: personagens, objetos e o mapa.
Entrada: *personagem*: o personagem atuante no turno.
Saída: *objetosEncontrados*: lista dos objetos identificados.

```
objetos ← mundo.objetos;
objetosLocais ← itens de objetos no mesmo local de personagem;
para todo objeto em objetosLocais faça
  cE ← personagem.Obv(objeto, "static"); {Componentes estáticos observados}
  cD ← personagem.Obv(objeto, "dynamic"); {Componentes dinâmicos observados}
  se (cE.tamanho > 0 ou cD.tamanho() > 0) então
    modelos ← personagem.ModelosSimilares(cE, cD); {Em ordem de similaridade}
    similar ← personagem.ObjetoSimilar(objeto, cE, cD); {Reconhecendo objeto}

    se (similar != nulo) então {objeto reconhecido}
      similar.AtualizaID(objeto.Id());
      similar.AtualizaLocalização(personagem.Localização);
      se (similar.cE != cE ou similar.cD != cD) então
        similar.AtualizaComponentes(cE, cD);
        para todo modelo em modelos faça
          similar.Atributos ← modelo.Atributos;
      se (último modelo.Nome de modelos != null e modelo.Nome não estiver em
      objetosEncontrados) então
        similar.Nome ←último modelo.Nome; {Nome do modelo mais similar}
      senão
        similar.Nome ← personagem.NomeIncognito();
      objetosEncontrados.Adiciona(similar);

    senão {objeto não reconhecido}
      novoObjeto = nova instância de objeto;
      novoObjeto.AtualizaComponentes(cE, CD);
      novoObjeto.AtualizaID(objeto.Id());
      novoObjeto.AtualizaLocalização(personagem.Localização);
      para todo modelo em modelos faça
        novoObjeto.Atributos ← modelo.Atributos;
      se (último modelo.Nome de modelos != nulo e modelo.Nome não estiver em
      objetosEncontrados) então
        novoObjeto.Nome ←último modelo.Nome;
      senão
        novoObjeto.Nome ← personagem.NomeIncognito();
      objetosEncontrados.Adiciona(novoObjeto);
      personagem.AdicionaObjeto(novoObjeto);
      se (novoObjeto é passagem de dois lugares) então
        personagem.AtualizaMapa(lugar ligado por novoObjeto);

retorna (objetosEncontrados);
```

incorretos. Outra possibilidade é o uso proposital de padrões que descrevam atributos erroneamente. O autor pode utilizar este recurso para descrever personagens que não entendam corretamente as propriedades que certos padrões de objetos devem ter.

4.2.2 Tratamento de Erros

Como consequência de observações e interpretações errôneas, os personagens podem interpretar o mesmo objeto visto em lugares diferentes como objetos diferentes, interpretar um objeto ou personagem como outro visto anteriormente, ou associá-los a padrões incorretos. Isso levará um personagem a criar planos cujas ações possam conter parâmetros incorretos, isto é, ações cujos elementos participantes na visão do personagem não condizem com os elementos participantes reais.

Para que estas ações possam ser devidamente executadas o sistema deve antes verificar se suas precondições são válidas. Para este propósito, o sistema se utiliza dos IDs gerados automaticamente para cada elemento do mundo e que são passados durante o processo de percepção. Sempre que um personagem tenta executar uma ação, o sistema troca seus parâmetros com os correspondentes reais através de seus IDs e re-analisa sua viabilidade.

Supondo que, em um conto de fadas, um dragão tenha por objetivo raptar a princesa, mas sem conhecer exatamente a sua aparência, ele identifica outra personagem como a desejada e tenta executar a ação "amedrontar" (detalhada abaixo). Esta ação exige como parâmetros um lugar, uma vítima e um amedrontador e, como precondições o amedrontador e a vítima estarem no mesmo lugar, e a vítima ter um valor de coragem inferior a 10. Analisando que o dragão tenta executar a ação usando o parâmetro "princesa" como vítima, o sistema verifica: qual é o ID do personagem "princesa" na base de dados do dragão, e qual é o verdadeiro personagem que possui o mesmo ID na base de dados do mundo. Então ele troca o parâmetro "princesa" pelo nome do personagem encontrado (que pode até mesmo ser a própria princesa). Essa mudança é feita para cada parâmetro da ação e então o sistema verifica se as precondições ainda são válidas. Em caso positivo, os efeitos da ação são aplicados no ambiente para os parâmetros reais, e na base de dados do dragão para os seus próprios parâmetros.

```
(:action scare
    :parameters (?frightener ?victim - character ?l - location)
    :preconditions (and(at l frightener)(at l victim)(<(courage victim)10))
    :effects (and (decrease (defense victim) 3))
)
```

Ademais, todos os personagens que estiverem na mesma localização do dragão atualizarão suas bases de dados com as precondições e efeitos da ação para todos os parâmetros que envolvam elementos que eles já conheciam usando suas próprias visões de mundo, ou seja, se eles tiverem visto a personagem usada como vítima antes da ação acontecer, eles atualizarão seus conhecimentos a respeito dessa personagem.

No Algoritmo 4.2 é apresentado parte das instruções utilizadas na execução de uma ação, com as quais, antes dos efeitos serem aplicados, é realizada uma verificação das precondições e a passagem dos parâmetros para seus correspondentes reais, isto é, para o ponto de vista do mundo. Antes da atualização da base de conhecimento dos personagens presentes, é realizada a passagem dos parâmetros para os seus pontos de vista.

Caso as precondições não sejam mais válidas, a ação é cancelada e o dragão deve tentar entender por quê. Sempre que uma ação for cancelada, seu executor passará a acreditar que todas suas precondições são falsas e tentará mudar seus valores em sua base de conhecimento. Esta mudança é orientada pelo tipo do valor do atributo a ser corrigido: se for booleano, então seu valor é invertido, se for numérico ou parte de uma lista numérica, então seu valor é corrigido com o valor verdadeiro, se for uma lista, então o valor usado na ação é removido da lista (ou adicionado, se o valor não fazia parte dela). No entanto, como o personagem não é capaz de testar se as precondições são realmente falsas, para evitar o acontecimento de becos-sem-saída, o sistema não permite que o personagem mude valores que estejam corretos. Ações incorretas também são incorporadas à história, mas com a indicação de que não foram devidamente efetivadas e o personagem apenas tentou executar a ação.

Se o dragão falhar ao executar a ação de amedrontar, então ele irá tentar atualizar os seus conhecimentos com as seguintes operações:

- Remoção da informação sobre sua própria localização, o que não seria executado, pois um personagem não pode estar errado sobre sua própria localização.
- Remoção da informação sobre a localização da vítima, o que seria executado apenas se a informação que o dragão possuir sobre a posição da vítima realmente estiver errada.
- Troca do valor de coragem da vítima, o que também seria executado apenas se a informação que o dragão possuir sobre a coragem da vítima realmente estiver errada.

É importante enfatizar que o dragão não deixará de pensar que a vítima em questão

Algoritmo 4.2 Parte do algoritmo de execução de uma ação.

Entrada: *mundo*: personagens, objetos e o mapa.
Entrada: *ação*: ação com parâmetros no ponto de vista do personagem atuante
Entrada: *personagemAtuante*: personagem que tenta realizar a *ação*
Saída: *resultado*: booleano que indica se a ação pôde ser executada

```
{Ao início, o sistema verifica se as precondições são realmente válidas}
se (!mundo.AnalisarPrecondições(ação, personagem)) então
  retorna (falso);
personagensPresentes ← mundo.personagens na localização de personagemAtuante;
para todo personagem p em personagensPresentes faça
  p.confirmaPrecondições(ao, personagemAtuante);

predicados ← ação.efeitos; {Exemplo de predicado: (hold chapeuzinho cesta)}
para todo predicado em ação faça
  {trata os predicados que dizem respeito a um personagem, como o exemplo acima}
  se (predicado é uma propriedade de personagem) então
    nomePr ← argumento1 de predicado;
    {nomePr é o nome do personagem referenciado no predicado (como Chapeuzinho)}
    personagemID ← ID do nomePr em personagemAtuante;
    personagemReferenciado ← personagem com personagemID em mundo;
    para todo (argumento de predicado) faça
      ID ← ID de argumento em personagemAtuante;
      argumentoVerdadeiro ← nome do elemento de ID em mundo;
      argumentosVerdadeiros.Adiciona(argumentoVerdadeiro);
    atributo ← atributo de personagemReferenciado com mesmo nome do predicado;
    atributo.Atualiza(argumentosVerdadeiros);

    para todo personagem p em personagensPresentes faça
      personagemConhecido ← personagem conhecido de p com personagemID;
      se (personagemConhecido == nulo) então
        continua;
      para todo (argumento de predicado) faça
        {Mudança de ponto de vista}
        ID ← ID de argumento em personagemAtuante;
        argumentoInterpretado ← nome do elemento de ID em p;
        argumentosInterpretados.Adiciona(argumentoVerdadeiro);
      se argumentosInterpretados contiver nulo então
        {Caso em que há argumentos desconhecidos: um personagem presente pode
        não ter visto a cesta}
        continua;
      atributo ← atributo de personagemConhecido com mesmo nome do predicado;
      atributo.Atualiza(argumentosInterpretados);
  senão
    ...
retorna (verdadeiro);
```

não é a princesa; ele apenas poderá passar a acreditar que a princesa tem um valor de coragem maior do que ele esperava, ou que não está no local indicado.

Um detalhe importante relativo ao uso dos identificadores é a possibilidade de dois ou mais elementos serem guardados na base de dados de um personagem com o mesmo ID. Quando um personagem percebe um determinado elemento de ID 1, por exemplo, que já conhecia, mas não o identifica corretamente (acha que viu algo novo ou se confundiu com outra coisa que também já conhecia), ele cria uma nova instância em sua base de dados com o ID 1 (se acredita ter visto algo novo) ou altera uma instância existente (se tiver se confundido com algo que já conhecia) atualizando-a com o ID 1. Dessa forma este personagem passa a ter duas instâncias com o mesmo ID 1 em sua base de dados, e como é necessário a um personagem, atualizar o conhecimento que possui sobre um elemento com base no ID, o sistema deve se basear em outro critério para realizar a atualização de forma consistente. A Figura 4.7 apresenta um exemplo de como duas instâncias de elementos conhecidos podem ter o mesmo ID.

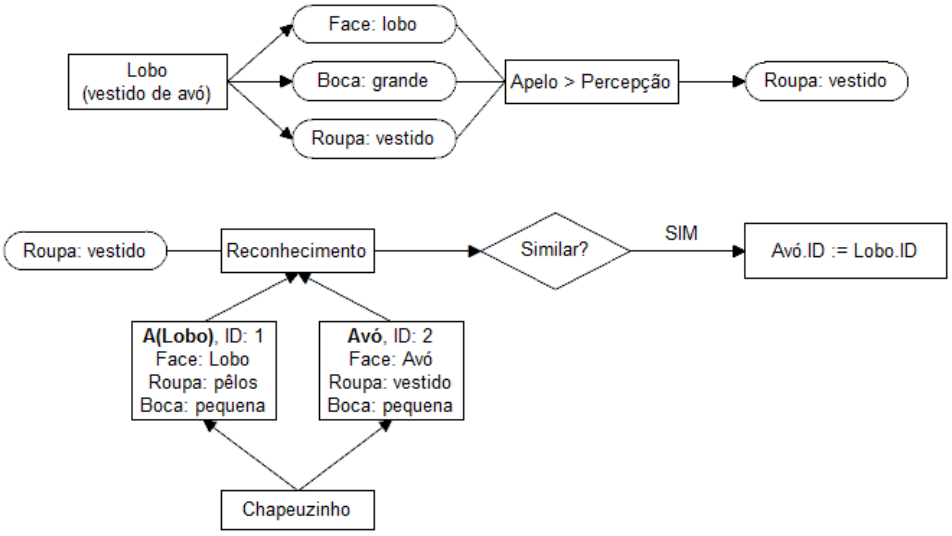

Figura 4.7: Exemplo de percepção com duplicação de ID

Sempre que o sistema tentar atualizar a base de conhecimento do personagem com um novo valor de atributo para um elemento de ID repetido, o sistema o faz com o elemento que se encontrar no mesmo local do personagem. Se nenhum deles estiver no mesmo local, então a atualização será feita no primeiro elemento encontrado com o mesmo ID.

4.3 Testes

Seguindo o caso proposto para testes do trabalho inicial, foi novamente escolhida a história da Chapeuzinho Vermelho para que possa ser feita uma análise do potencial das extensões inseridas. Neste caso, entretanto, o foco está na cena clímax característica da história que se baseia em um erro de percepção que pode ser usado pelo sistema para apresentar diferentes finais. A descrição de mundo foi alterada para seguir a nova forma de representação e abranger ações de todos os momentos da história original, incluindo a travessia da floresta.

4.3.1 Cenário

O mapa do mundo criado para esta história é composto por uma floresta, pelas casas da avó e da protagonista, seguindo o esquema mostrado na Figura 4.8.

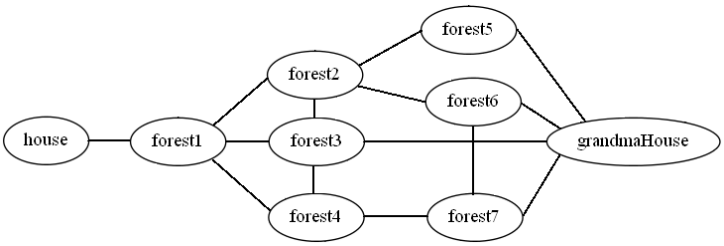

Figura 4.8: Mapa criado para a história da Chapeuzinho Vermelho

O único objeto de relativa importância utilizado neste contexto é a cesta de doces, que inicia a história na casa da protagonista, e deve ser entregue à sua avó. Dessa forma, como não há possibilidade de erros de interpretação entre objetos e estando o foco em erros de interpretação entre personagens, as descrições do único tipo e do único objeto utilizados foram feitas com poucos detalhes, que são explicitados a seguir:

```
Type: basket
dynamic: true

Attributes:
boolean pickable

Components:
dynamic: false
Name: Body

dynamic: true
Name: Content
```

```
Object basketofsweets
type: basket

Attributes:
pickable true

Components:
Body
Value: Wood
Appeal: 8

Content
Value: Sweets
Appeal: 8
```

Na versão do conto de fadas desenvolvida para este trabalho, foram utilizados os quatro personagens principais: Chapeuzinho, Avó, Lobo e Caçador com comportamentos, objetivos e modelos próprios. As tabelas 4.1 e 4.2 enumeram, respectivamente, os componentes e os atributos criados para os personagens tendo em vista o que foi considerado necessário para descrever o contexto no qual a história se desenvolve. Alguns deles não são usados por todos os personagens, mas devem constar na descrição de cada um.

Componente	**Mutabilidade**	**Descrição**
Cloth	dinâmico	Descrição da roupa que o personagem está usando.
Face	estático	Descrição da face do personagem.
Mouth	dinâmico	Descrição de alguma característica da boca do personagem.

Tabela 4.1: Componentes dos personagens

Atributo	**Tipo**	**Descrição**
dead	boolean	Indica se o personagem está morto.
defender	numeric	Indica quantas pessoas o personagem conheceu que poderiam defendê-lo.
defenseless	boolean	Determina se o personagem é indefeso.
disguised	boolean	Determina se o personagem está disfarçado.
evil	boolean	Indica se o personagem possui má índole.
fear	boolean	Indica se o personagem está sentindo medo.
inventory	list:location	Relaciona o personagem a um ou mais lugares que possam ser usados como seus inventários.
known	list:character	Indica quais são os personagens conhecidos.
lifepoints	numeric	Valor que descreve a quantidade de "pontos de vida" do personagem.
nearchr	list:character	Indica os personagens que estão próximos.
stamina	numeric	Valor que descreve a resistência do personagem.
stomach	list:location	Relaciona o personagem a um lugar que possa ser caracterizado como seu estômago.
strong	boolean	Indica se o personagem é capaz ou não de lutar.
talked	list:character	Indica com quem o personagem já conversou.
victim	numeric	Enumera a quantidade de vítimas do personagem.

Tabela 4.2: Atributos dos personagens

Chapeuzinho inicia a história em sua casa (representada no mapa como *house*), possui um valor médio de percepção, e tem como objetivo inicial entregar a cesta de doces para sua Avó. Para isso, ela precisará atravessar uma floresta que lhe é desconhecida. Seus

comportamentos são usados para refletir os conselhos de seus pais (e que são considerados a moral da história): ela deve se afastar de qualquer pessoa desconhecida e não deve ficar no mesmo lugar que uma pessoa que ela considere má, procurando neste último caso, ficar em um mesmo lugar que uma pessoa que possa protegê-la.

O Lobo inicia na entrada da floresta ($forest1$), próximo a casa de Chapeuzinho de tal forma que ela necessariamente o encontrará. Ele possui um valor alto de percepção, não tem objetivos iniciais e conhece a floresta inteira. Ele também conhece modelos de personagem, nem todos corretos, que indicam se uma pessoa é indefesa ou não. Seus comportamentos são: engolir qualquer pessoa indefesa que estiver próxima, aproximar-se de qualquer pessoa indefesa e "conversar" (executar uma ação que simboliza uma conversa) com qualquer pessoa que não seja indefesa.

O Caçador inicia a história no interior da floresta (em $forest2$). Ele possui um valor médio de percepção, não tem objetivos iniciais e conhece alguns pontos da floresta. Seu único comportamento é o de matar qualquer um que tenha má índole. A Avó tem como posição inicial sua própria casa ($grandmaHouse$), possui um valor baixo de percepção, não tem objetivos iniciais e possui comportamentos semelhantes aos da neta. No Apêndice A encontram-se as descrições completas de todos os elementos do mundo virtual.

O sistema não é capaz de executar diálogos ou permitir aos personagens que planejem suas próprias ações antecipando as ações dos demais. Para simular o diálogo entre Chapeuzinho e Lobo foi usado um evento para que ele passasse a conhecer a Avó após conversar com a Chapeuzinho e, para simular o disfarce do Lobo, foi usado um evento para mudar dois de seus componentes depois de engolir a avó da protagonista.

4.3.2 Resultados

A reprodução do cenário seguindo estas características tem por objetivo criar as condições necessárias para que uma sequência de ações que se assemelhe à história original possa ser gerada pelo sistema, porém com detalhes suficientes para permitir ao sistema variar o desenvolvimento e a conclusão de forma coerente com os personagens envolvidos. Nos testes realizados, foi possível obter uma história semelhante e quatro variações de conclusão. Na Figura 4.9 são apresentadas as ações da história considerada próxima à original (com texto adaptado) e uma análise explicitando os pontos onde erros de percepção influenciaram a atitude dos personagens.

A história original, por si só, possui versões variadas, que incluem diferentes motivos

1 Chapeuzinho Vermelho identifica e pega a cesta de doces.
2 Chapeuzinho Vermelho segue para Floresta1 e vê um personagem desconhecido.
3 O Lobo vê uma personagem desconhecida e passa a saber de sua avó.
4 Chapeuzinho vai para Floresta 4.
5 O Lobo vai para Floresta 3.
6 Chapeuzinho vai para Floresta 1.
7 O Lobo chega na casa da avó da Chapeuzinho e a encontra.
8 A avó da Chapeuzinho vê um personagem desconhecido, identifica como mau e passa a ter medo.
9 A Chapeuzinho vai para Floresta 2, encontra o Caçador e considera como um protetor.
10 O Lobo se aproxima da avó.
11 A Chapeuzinho segue para Floresta 3.
12 O Lobo engole a avó, se disfarça como ela e passa a mostrar a boca.
13 A Chapeuzinho descansa.
14 A Chapeuzinho chega na casa de sua avó.
15 A Chapeuzinho identifica o personagem que tinha visto, com má índole e passa a ter medo.
16 O Lobo reconhece a personagem que tinha visto antes.
17 A Chapeuzinho corre para fora da casa da avó e o Lobo percebe que ela é indefesa.
18 O Lobo vai para fora da casa da avó.
19 A Chapeuzinho corre para Floresta 2.
20 O Lobo vai para a Floresta 2.
21 O Caçador reconhece o Lobo e o mata abrindo seu estômago.
22 A Chapeuzinho se acalma.
23 A avó sai do estômago do Lobo, vê que está morto e passa a considerar o Caçador como um protetor.
24 A Chapeuzinho se aproxima da avó.
25 A Chapeuzinho entrega a cesta de doces para sua avó.

Figura 4.9: História gerada automaticamente pelo sistema

que explicam por quê o Lobo não ataca a protagonista logo no primeiro encontro após descobrir o que ela planeja fazer. Nesse caso decidiu-se por inserir, na base de conhecimentos do Lobo, um modelo incorreto que identifica a Chapeuzinho como um personagem não vulnerável, o que, em conjunto com seu comportamento de se aproximar apenas de quem ele considere vulnerável o impedem de atacá-la. A Chapeuzinho, por sua vez, possui dois modelos para analisar um personagem cujo componente "Face" seja característico de um lobo e apenas um deles o caracteriza como uma ameaça. O Lobo inicia a história com o componente para o qual a Chapeuzinho não o considera uma ameaça e por isso ela não decide fugir.

Como o Lobo possui o comportamento de conversar com qualquer pessoa que ele não considere vulnerável, ele realiza a ação "conversar", que por conseguinte ativa o evento

que faz o Lobo tomar conhecimento da existência da avó da Chapeuzinho, o que ativa o seu comportamento de tentar se aproximar de qualquer pessoa vulnerável.

Enquanto a Chapeuzinho deve "explorar" a floresta por não saber como chegar a casa de sua avó, o Lobo segue até o local pelo menor caminho possível e consegue se aproximar da personagem antes que a protagonista apareça. Essa aproximação ativa o seu comportamento de engolir qualquer pessoa que não seja forte e que esteja próxima, o que é feito em seguida. Neste ponto, o evento de disfarce é ativado e são feitas algumas trocas em seus componentes. Seu componente "Cloth" passa a ter o mesmo valor da personagem avó e seu componente "Mouth" passa a ter o valor "Big", usado para simbolizar uma das mudanças que a protagonista estranha na história original. Durante a travessia, a Chapeuzinho conhece o caçador, o que lhe serve para saber a localização de uma pessoa confiável.

O ponto-chave da história ocorre quando a protagonista chega à casa de sua avó, onde se encontra o Lobo disfarçado. Em seu processo de percepção, a Chapeuzinho recebe os componentes usados para descrever o Lobo, dentre os quais dois deles são dinâmicos e um deles é estático. Os componentes dinâmicos foram trocados durante o evento de disfarce, sendo que um deles é usado para tentar enganá-la, enquanto que o componente estático permanece inalterado. Neste caso, todos os componentes foram percebidos pela Chapeuzinho e, como os componentes estáticos têm prioridade no reconhecimento de elementos, ela identifica o personagem como sendo o mesmo que ela tinha visto antes. No entanto, como houve uma troca em dois dos componentes percebidos, ela verifica quais modelos estão de acordo com a nova aparência do personagem. Dado o novo valor do componente "Mouth", a personagem assume que o Lobo é uma ameaça.

Dois comportamentos foram definidos para a Chapeuzinho caso ela se encontre com um personagem de má índole: ir para perto de alguém que possa ajudá-la, caso ela conheça e ir para um lugar diferente do atual caso contrário, sendo que em ambos os comportamentos ela também deve passar a ter medo. Tendo conhecido o Caçador na floresta, a personagem passa a ter como objetivo chegar no último lugar em que o viu, mas por estar com medo, ao invés de usar a ação "mover" ela executa a ação "fugir", cujas precondições incluem ser um personagem vulnerável. Dessa forma, quando ela foge da casa, o Lobo descobre que a Chapeuzinho é uma personagem vulnerável e por isso passa a perseguí-la.

Quando a Chapeuzinho e o Lobo chegam ao local onde se encontra o Caçador, este também reconhece todos os componentes do Lobo e, por possuir um modelo que identifica

qualquer lobo como uma ameaça, ativa seu comportamento de matar qualquer personagem de má índole. Assim, o Caçador executa a ação "matar" que além de causar a morte do Lobo, também cria uma ligação entre o lugar atual e o estômago do antagonista. Feito isso, a avó da personagem principal sai do estômago do antagonista e finalmente recebe a cesta de doces da neta.

Duas variações imediatas obtidas para esta história são resultado da possibilidade da protagonista encontrar ou não o Caçador e reconhecer ou não o Lobo. Não encontrando o caçador e reconhecendo o Lobo, o sistema gerou uma história onde a Chapeuzinho tenta fugir, mas, por se cansar mais rápido que o Lobo, acaba sendo pega pelo personagem. Por outro lado, independente de encontrar o Caçador, não reconhecendo o Lobo o sistema gerou uma história onde a Chapeuzinho também terminou engolida pelo personagem por se aproximar dele para entregar os doces acreditando se tratar de sua avó.

O uso de percepções também resultou em um interessante efeito colateral: em uma das histórias geradas o Caçador não foi capaz de perceber o disfarce do Lobo. Como o Lobo tem o comportamento de conversar com quem ele não considere vulnerável, um de seus turnos foi perdido para conversar com o Caçador. No turno seguinte a Chapeuzinho conseguiu se distanciar do perseguidor, que em seguida não foi capaz de encontrá-la. Para obter mais uma variação partindo deste final, o comportamento do Lobo foi modificado para conversar com quem ele acreditasse não ser vulnerável e nem forte, dessa forma ao invés do Lobo perder um turno conversando com o Caçador, ele executou a ação de engolir a Chapeuzinho frente ao Caçador. Como a ação executada pelo Lobo tem como precondição o executor ser um personagem de má índole, o Caçador assumiu a que a suposta Avó era uma ameaça e matou o Lobo. A história é finalizada com a Chapeuzinho entregando a cesta de doces para sua avó depois de ambas saírem da barriga do Lobo.

Para as mesmas configurações do computador utilizado no teste presente no Capítulo 3, foram necessários em média 32 segundos para gerar cada história apresentada nesta seção.

4.4 Discussão

A história da Chapeuzinho Vermelho não pôde ser completamente reproduzida apenas pelos comportamentos e processos de raciocínio dos personagens criados, devido aos pontos em que, na história original, os personagens conversam e o Lobo se disfarça. Para que o sistema fosse capaz de concretizar estas atuações, foi necessário uso de dois eventos cujos

efeitos pudessem reproduzir as consequências destas atuações. Porém, como a mecânica empregada na avaliação de eventos foi criada para reproduzir regras que representem a forma como o mundo reage, os eventos são descritos com precondições feitas para serem atendidas em estados diversos. Como neste caso era preciso que em apenas dois momentos específicos os eventos fossem ativados, foi necessária a criação de dois atributos que controlassem os momentos em que eles fossem executados.

Entretanto, o modelo desenvolvido para simulação de percepções viabilizou a atuação relativa ao erro de reconhecimento feito pela personagem principal sem a necessidade de um evento pré-estabelecido, o que torna sua atuação mais natural e dá margem ao sistema para criar variações relativas a esse reconhecimento. Isso foi demonstrado pelos casos onde onde o próprio Caçador fez uma associação incorreta e agiu de acordo, gerando mais duas versões para a conclusão.

Também foram viabilizadas atuações como a inicial onde a protagonista e o Lobo não reagiram da maneira que supostamente seria a mais correta: a Chapeuzinho não fugiu e o Lobo não tentou atacá-la. As razões pela qual o antagonista não reproduz esse comportamento tem variado entre versões diferentes deste conto de fadas, mas a Chapeuzinho de fato não foge do Lobo justamente por não considerá-lo um perigo no início. Os dois modelos criados para a personagem podem ser interpretados como a sua própria ingenuidade em julgar o outro personagem, contrários ao modelo mais geral usado pelo Caçador que considera qualquer lobo perigoso. Os modelos são usados para simular a tentativa de um personagem de interpretar os atributos dos elementos do mundo que ele não conhece, o que representa seus preconceitos e nível de experiência.

Além dos desenvolvimentos citados, também foi gerado um caso onde Chapeuzinho e Lobo chegaram na casa da Avó ao mesmo tempo pelo mesmo caminho, no qual o Lobo terminou a história engolindo ambas. O fato de Chapeuzinho e Lobo seguirem para o final da história lado a lado sem maiores reações pode ser considerado incoerente, porém como mencionado, ambos mantinham conhecimentos incorretos que os impediam de agir de forma mais direta (como o Lobo engolir a Chapeuzinho imediatamente), o que mantém a coerência de seus comportamentos, embora essa possibilidade pudesse ser usada como forma de gerar novas variações interessantes.

A definição dos intervalos dos valores de percepção e aleatoriedade, bem como dos valores de percepção de cada personagem influenciam diretamente nas variações geradas pelo sistema. Os intervalos foram criados empiricamente, tendo por base sua divisão em intervalos menores que qualificassem a dificuldade de percepção de um componente e o

nível de percepção de um personagem, por exemplo componentes com valores de 1 a 3 de apelo são vistos com dificuldade, apenas por personagens com percepção entre 7 e 9.

Chapeuzinho Vermelho pode ser considerada uma história simples e pequena, mesmo quando comparada com outros contos de fada como Cinderela ou Branca de Neve, por exemplo. Os cenários envolvidos se resumem à casa da avó, à casa da protagonista e à floresta, enquanto os personagens se resumem aos quatro apresentados acima, salvo variações que incluem os pais da protagonista. Por isso, os personagens atuam de acordo com uma quantidade razoavelmente pequena de regras e atributos, que o autor pode definir através de repetidos, porém poucos testes, feitos em um ciclo de ajustes e observação dos resultados. Entretanto, é necessário que ele conheça a linguagem PDDL para a definição destas regras de comportamento, das ações dos personagens e dos eventos.

Para histórias mais complexas ou longas, isto é, que possuam uma quantidade considerável de ações e, cujas ações sejam diversificadas, a construção dos personagens é mais propensa a erros, pois comportamentos característicos do início de uma história podem entrar em conflito com comportamentos característicos de uma etapa posterior e a definição de determinados atributos e ações que apóiem os comportamentos pode não ser intuitiva. Pequenas mudanças em objetos e lugares podem se propagar por toda uma história, como podem não causar mudança alguma. Isso pode dificultar o desenvolvimento esperado de uma história, mas pode permitir a emergência de uma quantidade maior de situações interessantes, ainda com o fator do uso de percepções.

Neste caso de teste, onde as consequências das percepções podem ser antecipadas pelo autor, a escolha dos níveis de percepção foi feita de forma a tornar possível a confusão entre Lobo e Avó. Dessa forma, foi dado à percepção da Chapeuzinho e do Caçador percepções com valores em torno dos apelos dos componentes do Lobo após o disfarce. Em um cenário mais complexo, com mais personagens, objetos, variações de componentes e maior mapa, a escolha dos níveis de percepção necessitaria de repetidos experimentos e ajustes, com os quais o autor tentaria adequar os personagens ao desenvolvimento mais interessante. Para simular cenários onde os objetos têm uma variação considerável em termos de apelo, isto é, há objetos explicitamente escondidos e personagens que dificilmente os veriam, seria mais interessante, por exemplo, aumentar os intervalos de percepção e aleatoriedade.

Capítulo 5

Conclusão

Nas seções seguintes, a dissertação é concluída pela apresentação das considerações finais, dos detalhes relativos às contribuições alcançadas e propostas de possíveis trabalhos futuros.

5.1 Considerações finais

A pesquisa apresentada nesta dissertação teve por objetivo a geração de histórias nas quais os personagens fossem capazes de executar ações erradas que viessem a enriquecer suas atuações, tornando-os mais críveis e viabilizando novas possibilidades de desenvolvimento. Para tal, foi verificado em primeiro lugar os efeitos da inserção deste tipo de comportamento, em conjunto com uma análise das necessidades envolvidas para a sua geração e tratamento automáticos. Em seguida foi desenvolvido um modelo de simulação de percepções capaz de gerar erros nas bases de conhecimentos dos personagens coerentes com suas visões parciais de mundo. Através desta simulação foi possível reproduzir a história Chapeuzinho Vermelho e variações resultantes de erros de percepção.

Foi possibilitada a criação de erros inseridos nos personagens diretamente sobre atributos de elementos que ele conhece, como acreditar incorretamente que um personagem é forte ou fraco, e sobre a identidade destes elementos, como acreditar que o personagem A é o personagem B. Por meio destes erros foi possível executar testes onde os personagens atuaram de forma diferente do que teriam feito caso fossem oniscientes. A dificuldade principal enfrentada nesta metodologia foi a manutenção do conhecimento que os personagens têm de mundo de forma coerente. Como cada personagem possui sua própria visão de mundo, o sistema teve que ser capaz de fazer traduções entre estas visões.

Para tratar este problema, foi proposto o uso de um identificador único que se mantém nas cópias que os personagens possuem dos elementos que eles conhecem. Sempre que um personagem tenta executar uma ação, o sistema verifica se os elementos envolvidos realmente estão de acordo com as precondições. Caso não estejam, a ação é cancelada. A vantagem de se utilizar essa abordagem é a possibilidade de um personagem de executar ações com parâmetros trocados sem para isso tornar inconsistente a execução do sistema, e não necessariamente precisar de correções. Uma informação que um personagem acredita estar correta, mesmo estando errada, pode permanecer como conhecimento do personagem, sem alterações, mesmo após a execução de uma ação que dependa dela.

Entretanto, a metodologia de cancelamento trata corretamente ações que envolvam apenas os elementos que foram confundidos e seus correspondentes reais, mas não aquelas que possam ser aplicadas a um terceiro elemento.

Para exemplificar o problema, supondo que em uma história qualquer, um personagem deseje ativar uma câmara de gás para matar um personagem A que ele acredita estar em seu interior. Se a câmara de gás exige que uma pessoa realmente esteja em seu interior para ser ativada, o sistema irá verificar quem é o personagem A retratado na ação para saber se ele se encontra no interior da câmara. Se o verdadeiro personagem estiver na câmara, então a ação é executada, caso contrário é cancelada. O problema decorre do fato de que: se o personagem real não estiver na câmara, mas existir uma outra pessoa em seu interior, a ação deveria ser executada, no entanto é cancelada. Seria possível ao sistema verificar, antes de cancelar uma ação, se existem outras condições com as quais ela poderia ser executada. Entretanto, esse método forçaria a criação e execução de ações com parâmetros diferentes, que não necessariamente estariam de acordo com a original. Quando um personagem tentasse atravessar uma porta que estivesse trancada, o sistema faria com que, por meio deste método, o personagem atravessasse uma outra porta que estivesse aberta, o que representa uma ação diferente. Dessa forma, optou-se por manter o cancelamento destas ações.

5.2 Análise das Contribuições

Um modelo para a descrição de personagens e objetos baseado em suas características físicas.

Para viabilizar a simulação proposta, foi utilizada uma descrição de mundo virtual onde a aparência física dos objetos e dos personagens também é retratada. Foram adicio-

nados às suas descrições conjuntos de um novo elemento denominado componente, o qual é usado para explicitar o quanto o personagem ou objeto se destaca no ambiente e como eles podem ser interpretados quando são vistos.

Um processo de percepção dividido em uma etapa de observação, que distingue o que os personagens são capazes de perceber e uma etapa de interpretação que determina as expectativas de um personagem a respeito de elementos percebidos.

Foi desenvolvido um processo de observação que, a partir de um valor de percepção dado aos personagens e um valor de apelo dado aos componentes, determina quais características físicas os personagens são capazes de discernir dos elementos de mundo que eles vêm. Partindo dessa visão diminuída, foi criado um processo de interpretação que associa características físicas a padrões conhecidos, usados pelos personagens para tentar adivinhar quais são as propriedades dos elementos que eles desconhecem. A associação incorreta é viabilizada propositalmente de forma a gerar informações erradas.

Um tratamento de erros para a análise e execução dos efeitos de ações realizadas com base em informações erradas.

A cada personagem, objeto e lugar do mundo é dado um identificador usado pelo sistema para discernir os parâmetros usados pelos personagens em suas ações dos parâmetros reais. Por conseguinte, o sistema realiza apenas ações que mesmo executadas com parâmetros errados, têm suas precondições atendidas, e faz o personagem aprender com ações que tenham falhado. Todas as mudanças causadas por eventos ou ações são percebidas pelos personagens que estiverem nos mesmos locais onde a mudança ocorreu seguindo suas próprias visões de mundo

5.3 Trabalhos futuros

Os processos de planejamento e percepção são trabalhados separadamente, de tal forma que aos personagens não é possível raciocinar considerando aspectos da própria percepção. Uma proposta de trabalho futuro seria desenvolver um algoritmo de planejamento que se utilize da percepção e também permita aos personagens antecipar as ações uns dos outros. Usando nos modelos de personagens não só características, mas também objetivos, um personagem poderia se utilizar de um modelo para tentar analisar quais as intenções de outro personagem. Isso, em conjunto com um planejamento perceptivo, permitiria a execução natural de ações como disfarce. Outra extensão imediata, também relacionada

à naturalidade dos personagens, é a utilização de ações durativas removendo a divisão da linha do tempo em turnos, o que exigiria o uso de um planejador com as capacidades necessárias para tanto, como em [22], e a adaptação do processo de percepção para análise contínua do ambiente ou discreta executada sempre que uma mudança for executada. Isso permitiria a execução de ações conjuntas e interdependentes.

Para que esta simulação pudesse ser realizada, foram feitas mudanças na representação dos elementos que compõem um mundo virtual, de forma a englobar também suas descrições físicas. Estas descrições foram usadas para que os personagens pudessem perceber o mundo como se utilizassem o sentido da visão. Uma adição interessante ao sistema de geração seria viabilizar o uso de outros sentidos que contribuíssem para uma análise mais detalhada ou mesmo errônea do ambiente. O uso do sentido da audição, por exemplo, poderia ser usado para que os personagens percebessem elementos que estivessem em lugares próximos ao que ele se encontra. Para isso seria necessário complementar a descrição dos componentes com novas características que pudessem ser usadas para simbolizar a reprodução de sons. Como sons são resultado de interações entre elementos diferentes, seria interessante descrevê-los como um elemento extra com componentes próprios, que seriam resultado de combinações entre os componentes dos elementos usados para gerá-los. Conhecendo diversos modelos de elementos com componentes semelhantes, os personagens poderiam se confundir tentando descobrir quais deles teriam sido usados para gerar os sons ouvidos.

Ações canceladas são inseridas como parte da história, pois mesmo não sendo executadas são parte das atitudes dos personagens, entretanto são inseridas sem explicitação do motivo que levou à falha. Um extensão interessante a esta pesquisa seria a manutenção das percepções de cada personagem, de forma a tornar possível o rastreio do que levou um personagem a tomar uma dada informação como correta e a tentar executar uma ação falha. Assim passa a ser possível a apresentação de uma ação cancelada com as devidas explicações, que podem não ter sido notadas pelo usuário que estiver acompanhando a história gerada.

Nesta pesquisa, optou-se por fazer com que os personagens que não participassem de uma ação ou evento, mas estivessem no mesmo lugar onde ela ocorre sejam capazes de atualizar seus conhecimentos com as precondições e efeitos do que tiver sido realizado. Para isso, cada parâmetro da mudança realizada, seja evento ou ação, deve ser alterada para que se enquadre na base de conhecimento de cada personagem que a viu, o que é feito através dos identificadores. Como mais de um elemento pode ter o mesmo ID dentro

de uma base de conhecimento, uma outra forma de realizar o processo de análise de uma ação no ponto de vista de um personagem seria fazer uma busca pelo elemento que possua o ID procurado e esteja de acordo com o maior número de precondições da ação realizada ou de acordo com precondições mais relevantes. Em suma, ao invés de optar por atualizar o elemento de ID correspondente à ação que estiver no mesmo local do personagem, como é feito por este sistema, a atualização poderia ser feita no elemento de mesmo ID cujos atributos estejam mais próximos das precondições da ação.

Outra mudança interessante relacionada ao processo perceptivo seria inserir a possibilidade de interpretação baseada não só com base nos componentes, como também dos próprios atributos. Em uma das variações geradas para a história Chapeuzinho Vermelho, o Caçador mata o Lobo mesmo acreditando se tratar da avó por tê-la considerado uma ameaça após vê-la engolindo a protagonista. Com o auxílio deste método, o Caçador poderia ter percebido que aquele personagem não era a avó e sim o Lobo, pois ele era o único personagem mal da história. Para essa mudança, seria necessário o uso de um método no processo de interpretação a ser usado pelos personagens para que eles fossem capazes de, após presenciar uma ação, analisar se o mais correto é atualizar o que ele sabe sobre um elemento ou acreditar que este elemento foi incorretamente interpretado.

Por fim, apesar da componente randômica utilizada na observação, a variação de percepção foi pouco explorada pelo sistema. Seria interessante a utilização de mecanismos vinculados à história que variassem o valor de percepção de um personagem durante a sua geração. A percepção poderia ser alterada em função de valores de tensão, simbolizando a influência das emoções no poder de observação de um personagem. Outra possibilidade significativa é dar ao usuário a capacidade de interagir com o sistema alterando estes valores de percepção, para a geração de histórias que sejam mais adequadas aos seus interesses.

Referências

[1] Albuquerque, A. Um sistema de storytelling textual para gerar histórias que despertem surpresa, suspense e curiosidade. Dissertação de Mestrado, Universidade Federal de Santa Maria, Santa Maria, RS, Brasil, Setembro 2011.

[2] Aylett, R.; Louchart, S.; Dias, J.; Paiva, A.; Vala, M. Fearnot!–an experiment in emergent narrative. In *Intelligent Virtual Agents* (2005), Springer, pp. 305–316.

[3] Bal, M. *Narratology: Introduction to the theory of narrative*, 2 ed. University of Toronto Pr, 1997.

[4] Barros, L.; Musse, S. Towards consistency in interactive storytelling: Tension arcs and dead-ends. *Computers in Entertainment (CIE) 6*, 3 (2008), 1–17.

[5] Cavazza, M.; Charles, F.; Mead, S. Character-based interactive storytelling. *IEEE Intelligent Systems* (2002), 17–24.

[6] Ciarlini, A.; Casanova, M.; Furtado, A.; Veloso, P. Modeling interactive storytelling genres as application domains. *Journal of Intelligent Information Systems 35*, 3 (2010), 347–381.

[7] Ciarlini, A.; Pozzer, C.; Furtado, A.; Feijó, B. A logic-based tool for interactive generation and dramatization of stories. In *Proceedings of the 2005 ACM SIGCHI International Conference on Advances in computer entertainment technology* (2005), ACM, pp. 133–140.

[8] Damiano, R.; Lombardo, V. Value-driven characters for storytelling and drama. *AI* IA 2009: Emergent Perspectives in Artificial Intelligence* (2009), 436–445.

[9] Dória, T.; Ciarlini, A.; Andreatta, A. A nondeterministic model for controlling the dramatization of interactive stories. In *Proceeding of the 2nd ACM international workshop on Story representation, mechanism and context* (2008), ACM, pp. 21–26.

[10] Ghallab, M.; Howe, A.; Christianson, D.; McDermott, D.; Ram, A.; Veloso, M.; Weld, D.; Wilkins, D. Pddl—the planning domain definition language. *AIPS98 planning committee 78*, 4 (1998), 1–27.

[11] Ghallab, M.; Nau, D.; Traverso, P. *Authomated Planning: Theory and Practice*, 1 ed. Morgan Kaufmann Publishers, Amsterdam, 2004.

[12] Karlsson, B. *A model and an interactive system for plot composition and adaptation, based on plan recognition and plan generation.* Tese de Doutorado, Pontifícia Universidade Católica do Rio de Janeiro.

[13] Kruizinga, E. Planning for character agents in automated storytelling. Dissertação de Mestrado, University of Twente, Enschede, Holanda, Outubro 2007.

[14] Lebowitz, M. Story-telling as planning and learning. *Poetics 14*, 6 (1985), 483–502.

[15] Lima, E. Um modelo de dramatização baseado em agentes cinematográficos autônomos para storytelling interativo. Dissertação de Mestrado, Universidade Federal de Santa Maria, Santa Maria, RS, Brasil, Março 2010.

[16] Mateas, M.; Stern, A. Façade: An experiment in building a fully-realized interactive drama. In *Game Developers Conference Game Design track* (2003), vol. 2, Citeseer.

[17] Meehan, J. Tale-spin. In *Inside Computer Understanding: Five Programs Plus Miniatures*, R. C. Schank and C. K. Riesbeck, Eds. Lawrence Erlbaum, 1981, pp. 197–226.

[18] Móra, M. *Um Modelo Formal e Executável de Agentes BDI.* Tese de Doutorado, Universidade Federal do Rio Grande do Sul, Porto Alegre, RS, Brasil, Setembro 1999.

[19] Passos, E.; Montenegro, A.; Clua, E.; Pozzer, C.; da Silva, F. Hierarchical pnf networks-a temporal model of events for the representation and dramatization of storytelling. In *Games and Digital Entertainment (SBGAMES), 2009 VIII Brazilian Symposium on* (2009), IEEE, pp. 175–184.

[20] Pizzi, D.; Cavazza, M. Affective storytelling based on characters' feelings. In *Proceedings of the AAAI Fall Symposium on Intelligent Narrative Technologies* (Arlington, Virginia, Novembro 2007).

[21] Porteous, J.; Cavazza, M.; Charles, F. Applying planning to interactive storytelling: Narrative control using state constraints. *ACM Transactions on Intelligent Systems and Technology (TIST) 1*, 2 (2010), 1–21.

[22] Porteous, J.; Teutenberg, J.; Charles, F.; Cavazza, M. Controlling narrative time in interactive storytelling. In *Proc. of 10th Int. Conf. on Autonomous Agents and MultiAgent Systems (AAMAS 2011)* (2011).

[23] Pozzer, C. *Um sistema para geração, interação e visualização 3D de histórias para TV interativa.* Tese de Doutorado, Pontifícia Universidade Católica do Rio de Janeiro, Rio de Janeiro, RJ, Brasil, Março 2005.

[24] Propp, V. *Morphology of the Folktale*, vol. 9. Univ of Texas Pr, 1968.

[25] Riedl, M. O.; Young, R. An intent-driven planner for multi-agent story generation. In *Proceedings of the Third International Joint Conference on Autonomous Agents and Multiagent Systems-Volume 1* (2004), IEEE Computer Society, pp. 186–193.

[26] Saigol, Z. A. Intelligent planning for autonomous underwater vehicles: Rsmg report 2. Tech. rep., School of Computer Science, University of Birmingham, 2007.

[27] Silva, F. Geração de enredos com planejamento não-determinístico em storytelling para tv interativa. Dissertação de Mestrado, Universidade Federal do Estado do Rio de Janeiro, Rio de Janeiro, RJ, Brasil, Setembro 2010.

[28] Su, W.; Pham, B.; Wardhani, A. Personality and emotion-based high-level control of affective story characters. *Visualization and Computer Graphics, IEEE Transactions on 13*, 2 (2007), 281–293.

[29] Swartjes, I. *Whose Story Is It Anyway? How Improv Informs Agency and Authorship of Emergent Narrative.* Tese de Doutorado, University of Twente, Enschede, Holanda, Maio 2010.

[30] Swartjes, I.; Theune, M. A fabula model for emergent narrative. *Technologies for Interactive Digital Storytelling and Entertainment* (2006), 49–60.

[31] Theune, M.; Rensen, S.; Akker, R.; Heylen, D.; Nijholt, A. Emotional characters for automatic plot creation. *Technologies for Interactive Digital Storytelling and Entertainment* (2004), 95–100.

[32] Theune, M.; Slabbers, N.; Hielkema, F. The narrator: Nlg for digital storytelling. In *Proceedings of the Eleventh European Workshop on Natural Language Generation* (2007), Association for Computational Linguistics, pp. 109–112.

APÊNDICE A – Descrição do Mundo Virtual

Neste apêndice são detalhados os elementos utilizados para compor o cenário apresentado como teste no Capítulo 4. As sequências a seguir mostram os arquivos criados para descrever: o tipo de objeto, o objeto, o modelo de objeto, o mapa, os atributos e componentes dos personagens, os modelos de personagens, os personagens e os eventos.

```
Type: basket
dynamic: true

Attributes:
boolean pickable

Components:
dynamic: false
Name: Body

dynamic: true
Name: Content
```

Figura A.1: Descrição do tipo de objeto cesta

```
Model BasketofSweets
name: basketofsweets

type: basket

Attributes:
pickable true

Components:
Body
Value: Wood

Content
Value: Sweets
```

Figura A.2: Descrição do modelo cesta de doces

```
Object basketofsweets
type: basket

Attributes:
pickable true

Components:
Body
Value: Wood
Appeal: 8

Content
Value: Sweets
Appeal: 8
```

Figura A.3: Descrição do objeto cesta de doces

```
Map Forest

Places: littleredhouse grandmahouse forest1 forest2 forest3 forest4 forest5
    forest6 forest7 littleredpocket grandmapocket wolfstomach grandmastomach

Place: littleredhouse
Objects: basketofsweets
Neighbors:
forest1


Place: forest1
Objects:
Neighbors:
littleredhouse
forest2
forest3
forest4


Place: forest2
Objects:
Neighbors:
forest1
forest3
forest5
forest6


Place: forest3
Objects:
Neighbors:
forest1
forest2
forest4
grandmahouse
```

```
Place: forest4
Objects:
Neighbors:
forest1
forest3
forest7


Place: forest5
Objects:
Neighbors:
forest2
grandmahouse


Place: forest6
Objects:
Neighbors:
forest2
forest7
grandmahouse


Place: forest7
Objects:
Neighbors:
forest4
forest6
grandmahouse


Place: grandmahouse
Objects:
Neighbors:
forest3
forest5
forest6
forest7


Place: littleredpocket
Objects:
Neighbors:


Place: grandmapocket
Objects:
Neighbors:


Place: wolfstomach
Objects:
Neighbors:


Place: grandmastomach
Objects:
Neighbors:
```

Figura A.4: Descrição do mapa

```
numeric stamina
numeric victim
numeric defender
numeric lifepoints
boolean fear
boolean evil
boolean strong
boolean dead
boolean defenseless
boolean desguised
list:location inventory
list:location stomach
list:character talked
list:character nearchr
list:character known

Components:
dynamic: false
Name: Face
Value: String

dynamic: true
Name: Mouth
Value: String

dynamic: true
Name: Cloth
Value: String
```

Figura A.5: Arquivo auxiliar de descrição de personagens

```
Model littlegirl

Attributes:
defenseless false
Components:
Face
Value: LittleGirl


Model goodwolf

Attributes:
evil false
Components:
Face
Value: Wolf
Mouth
Value: Little


Model badcharacter

Attributes:
evil true
Components:
Mouth
Value: Big
```

```
Model badwolf

Attributes:
dead false
lifepoints 10
evil true
stomach wolfstomach

Components:
Face
Value: Wolf
Mouth
Value: Big

Model deadWolf

Attributes:
dead true
lifepoints 0

Components:
Face
Value: Wolf
Cloth
Value: Pierced
Mouth
Value: Cut

Model wolf

Attributes:
evil true
dead false
stomach wolfstomach
lifepoints 10

Components:
Face
Value: Wolf

Model grandma

Attributes:
evil false
stomach grandmastomach

Components:
Cloth
Value: Dress

Model hunter

Attributes:
strong true
evil false

Components:
Face
Value: Hunter
Mouth
Value: Little
Cloth
Value: Hunter
```

Figura A.6: Descrição dos modelos de personagens

```
Character hunter

Attributes:

evil false
dead false
fear false
strong true
desguised false
defenseless false
stamina 10
victim 0
defender 0
lifepoints 10
inventory
stomach
talked
known
nearchr

Place: forest2

Vision: 5

KnowObjects:
KnowObjectModels:
KnowCharacters: grandma littlered
KnowCharacterModels: littlegirl wolf grandma deadWolf
KnowPlaces: forest1 forest2 forest3 forest4 forest5 forest6 forest7 grandmahouse

Components:
Face
Value: Hunter
Appeal: 7

Mouth
Value: Little
Appeal: 7

Cloth
Value: Hunter
Appeal: 8

Actions:

(:action move
       :parameters (?c - character ?from ?to - location)
       :precondition (and (actor ?c) (> (stamina ?c) 0) (att ?from ?c)
                     (near ?to ?from))
       :effect (and (not (att ?from ?c)) (att ?to ?c))
)

(:action killOpeningStomach
       :parameters (?c ?c2 - character ?l ?l2 - location)
       :precondition (and (actor ?c) (not (actor ?c2)) (att ?l ?c) (att ?l ?c2)
                     (stomach ?c2 ?l2))
       :effect (and (near ?l ?l2) (near ?l2 ?l) (decrease (lifepoints ?c2) 10))
)

(:action openStomach
       :parameters (?c ?c2 - character ?l ?l2 - location)
       :precondition (and (actor ?c) (not (actor ?c2)) (att ?l ?c) (att ?l ?c2)
                     (dead ?c2) (stomach ?c2 ?l2))
       :effect (and (near ?l ?l2) (near ?l2 ?l))
)
```

```
(:action pick
       :parameters (?c - character ?l1 ?l2 - location ?o - object ?b - basket)
       :precondition (and (actor ?c) (att ?l1 ?c) (in ?l1 ?o)
                          (inventory ?c ?l2) (pickable ?b))
       :effect (and (not (in ?l1 ?o)) (in ?l2 ?o))
)

(:action rest
       :parameters (?c - character ?l - location)
       :precondition (and (actor ?c) (att ?l ?c))
       :effect (and (increase (stamina ?c) 3))
)

Behaviors:

(:behavior killEvil
       :parameters (?c ?c2 - character ?l - location)
       :precondition (and (actor ?c) (not (actor ?c2)) (evil ?c2)
                          (not (dead ?c2)) (att ?l ?c) (att ?l ?c2))
       :goal (and (< (lifepoints ?c2) 1))
)
```

Figura A.7: Descrição do personagem Caçador

```
Character grandma

Attributes:

evil false
dead false
fear false
strong false
desguised false
defenseless true
stamina 1
victim 0
defender 0
lifepoints 3
inventory grandmapocket
known littlered
stomach
talked
nearchr

Place: grandmahouse

Vision: 4

KnowObjects:
KnowObjectModels:
KnowCharacters: littlered
KnowCharacterModels: wolf deadWolf
KnowPlaces: forest7 forest6 forest5 forest3 grandmahouse grandmapocket
```

```
Components:
Face
Value: Grandma
Appeal: 7

Mouth
Value: Little
Appeal: 7

Cloth
Value: Dress
Appeal: 8

Actions:

(:action move
       :parameters (?c - character ?from ?to - location)
       :precondition (and (actor ?c) (> (stamina ?c) 2) (att ?from ?c)
                     (near ?to ?from))
       :effect (and (not (att ?from ?c)) (att ?to ?c) (decrease (stamina ?c) 2))
)

(:action haveFear
       :parameters (?c ?c2 - character ?l - location)
       :precondition (and (actor ?c) (not (fear ?c)) (att ?l ?c) (att ?l ?c2)
                     (evil ?c2))
       :effect (and (fear ?c))
)

(:action pick
       :parameters (?c - character ?l1 ?l2 - location ?o - object)
       :precondition (and (actor ?c) (att ?l1 ?c) (in ?l1 ?o)
                     (inventory ?c ?l2))
       :effect (and (not (in ?l1 ?o)) (in ?l2 ?o))
)

(:action rest
       :parameters (?c - character ?l - location)
       :precondition (and (actor ?c) (att ?l ?c))
       :effect (and (increase (stamina ?c) 2))
)

Behaviors:

(:behavior getOutEvil
       :parameters (?c ?c2 - character ?l ?l2 - location)
       :precondition (and (actor ?c) (not (actor ?c2)) (att ?l2 ?c)
             (att ?l ?c2) (evil ?c2) (stomach ?c2 ?l2) (near ?l ?l2))
       :goal (and (not (att ?l2 ?c)))
)

(:behavior fearEvil
       :parameters (?c ?c2 - character ?l - location)
       :precondition (and (actor ?c) (att ?l ?c) (att ?l ?c2) (not (fear ?c))
                     (evil ?c2) (not (dead ?c2)))
       :goal (and (fear ?c))
)
```

```
(:behavior notNearUnknow
       :parameters (?c ?c2 - character)
       :precondition (and (actor ?c) (nearchr ?c ?c2) (not (known ?c ?c2))
                     (not (dead ?c2)))
       :goal (and (not (nearchr ?c ?c2)))
)

(:behavior runFromEviltoUnknow
       :parameters (?c ?c2 - character ?l - location)
       :precondition (and (actor ?c) (att ?l ?c) (fear ?c) (att ?l ?c2)
                     (evil ?c2) (not (dead ?c2)) (< (defender ?c)1))
       :goal (and (not (att ?l ?c)))
)

(:behavior runFromEviltoDefence
       :parameters (?c ?c2 ?c3 - character ?l ?l2 - location)
       :precondition (and (actor ?c) (att ?l ?c) (fear ?c) (att ?l ?c2)
                     (att ?l2 ?c3) (evil ?c2) (not (evil ?c3))
                     (strong ?c3) (not (dead ?c2)))
       :goal (and (att ?l2 ?c))
)
```

Figura A.8: Descrição da personagem Avó

```
Character wolf

Attributes:

evil true
dead false
fear false
strong true
desguised false
defenseless false
stamina 10
victim 0
defender 0
lifepoints 10
inventory
stomach wolfstomach
talked
known
nearchr

Place: forest1

Vision: 5

KnowObjects:
KnowObjectModels:
KnowCharacters:
KnowCharacterModels: littlegirl hunter
KnowPlaces: forest1 forest2 forest3 forest4 forest5 forest6 forest7
    grandmahouse wolfstomach
```

```
Components:
Face
Value: Wolf
Appeal: 7

Mouth
Value: Little
Appeal: 7

Cloth
Value: Bristle
Appeal: 8

Actions:

(:action move
       :parameters (?c - character ?from ?to - location)
       :precondition (and (actor ?c) (not (dead ?c)) (> (stamina ?c) 0)
                     (att ?from ?c) (near ?to ?from))
       :effect (and (not (att ?from ?c)) (decrease (stamina ?c) 1) (att ?to ?c))
)

(:action eat
       :parameters (?c ?c2 - character ?from ?to - location)
       :precondition (and (actor ?c) (not (actor ?c2)) (not (dead ?c))
                     (evil ?c) (att ?from ?c) (att ?from ?c2)
                     (nearchr ?c ?c2) (stomach ?c ?to))
       :effect (and (not (att ?from ?c2)) (not (nearchr ?c ?c2)) (not (nearchr
          ?c2 ?c)) (att ?to ?c2))
)

(:action getNearChr
       :parameters (?c ?c2 - character ?l - location)
       :precondition (and (actor ?c) (not (dead ?c)) (not (actor ?c2))
                     (att ?l ?c) (att ?l ?c2))
       :effect (and (nearchr ?c ?c2) (nearchr ?c2 ?c))
)

(:action talk
       :parameters (?c ?c2 - character ?l - location)
       :precondition (and (actor ?c) (not (dead ?c)) (not (actor ?c2))
                     (att ?l ?c) (att ?l ?c2))
       :effect (and (talked ?c ?c2) (talked ?c2 ?c))
)

(:action pick
       :parameters (?c - character ?l1 ?l2 - location ?o - object ?b - basket)
       :precondition (and (actor ?c) (not (dead ?c)) (att ?l1 ?c) (in ?l1 ?o)
                     (inventory ?c ?l2) (pickable ?b))
       :effect (and (not (in ?l1 ?o)) (in ?l2 ?o))
)

(:action rest
       :parameters (?c - character ?l - location)
       :precondition (and (actor ?c) (not (dead ?c)) (att ?l ?c))
       :effect (and (increase (stamina ?c) 3))
)
```

```
Behaviors:

(:behavior approximateVictim
       :parameters (?c ?c2 - character ?l - location)
       :precondition (and (actor ?c) (not (actor ?c2)) (not (nearchr ?c ?c2))
                     (defenseless ?c2) (stomach ?c ?l) (not (att ?l ?c2)))
       :goal (and (nearchr ?c ?c2))
)

(:behavior talktoVictim
       :parameters (?c ?c2 - character ?l ?l2 - location)
       :precondition (and (actor ?c) (not (actor ?c2)) (att ?l ?c) (att ?l ?c2)
                     (not (defenseless ?c2)) (not (strong ?c2))
                     (not (talked ?c ?c2)))
       :goal (and (talked ?c ?c2))
)

(:behavior eatVictim
       :parameters (?c ?c2 - character ?l ?l2 - location)
       :precondition (and (actor ?c) (not (actor ?c2)) (att ?l ?c) (att ?l ?c2)
                     (not (strong ?c2)) (stomach ?c ?l2) (nearchr ?c ?c2))
       :goal (and (att ?l2 ?c2))
)
```

Figura A.9: Descrição do personagem Lobo

```
Character littlered

Attributes:

fear false
dead false
evil false
strong false
desguised false
defenseless true
stamina 6
victim 0
defender 0
lifepoints 5
inventory littleredpocket
known grandma
stomach
talked
nearchr

Place: littleredhouse

Vision: 5

KnowObjects:
KnowObjectModels: standardbasket
KnowCharacters: grandma hunter
KnowCharacterModels: badwolf goodwolf deadWolf hunter badcharacter
KnowPlaces: littleredhouse littleredpocket grandmapocket
```

```
Components:
Face
Value: LittleGirl
Appeal: 7

Mouth
Value: Little
Appeal: 7

Cloth
Value: RedCape
Appeal: 8

Actions:

(:action move
       :parameters (?c - character ?from ?to - location)
       :precondition (and (actor ?c) (not (fear ?c)) (> (stamina ?c) 0)
                     (att ?from ?c) (near ?to ?from))
       :effect (and (not (att ?from ?c)) (att ?to ?c) (decrease (stamina ?c) 1))
)

(:action run
       :parameters (?c - character ?from ?to - location)
       :precondition (and (actor ?c) (fear ?c) (> (stamina ?c) 0)
                (defenseless ?c) (att ?from ?c) (near ?to ?from))
       :effect (and (not (att ?from ?c)) (att ?to ?c) (decrease (stamina ?c) 1))
)

(:action haveFear
       :parameters (?c ?c2 - character ?l - location)
       :precondition (and (actor ?c) (not (fear ?c)) (att ?l ?c) (att ?l ?c2)
                     (evil ?c2))
       :effect (and (fear ?c))
)

(:action calm
       :parameters (?c - character)
       :precondition (and (actor ?c) (fear ?c))
       :effect (and (not (fear ?c)))
)

(:action moveAway
       :parameters (?c ?c2 - character)
       :precondition (and (actor ?c) (> (stamina ?c) 3) (nearchr ?c ?c2))
       :effect (and (not (nearchr ?c ?c2)) (decrease (stamina ?c) 3)
                (not (nearchr ?c2 ?c)))
)

(:action getNearChr
       :parameters (?c ?c2 - character ?l - location)
       :precondition (and (actor ?c) (att ?l ?c) (att ?l ?c2))
       :effect (and (nearchr ?c ?c2) (nearchr ?c2 ?c))
)

(:action pick
       :parameters (?c - character ?l1 ?l2 - location ?b - basket)
       :precondition (and (actor ?c) (att ?l1 ?c) (in ?l1 ?b)
                     (inventory ?c ?l2) (pickable ?b))
       :effect (and (not (in ?l1 ?b)) (in ?l2 ?b))
)
```

```
(:action rest
       :parameters (?c - character ?l - location)
       :precondition (and (actor ?c) (att ?l ?c))
       :effect (and (increase (stamina ?c) 5))
)

(:action give
       :parameters (?c ?c2 - character ?l ?l1 ?l2 - location ?o - object)
       :precondition (and (actor ?c) (att ?l ?c) (att ?l ?c2) (in ?l1 ?o)
                (inventory ?c ?l1) (inventory ?c2 ?l2) (nearchr ?c ?c2))
       :effect (and (not (in ?l1 ?o)) (in ?l2 ?o))
)

Behaviors:

Initial:
(and (in littleredpocket basketofsweets))
(and (att grandmahouse littlered))
(and (in grandmapocket basketofsweets))


(:behavior fearEvil
       :parameters (?c ?c2 - character ?l - location)
       :precondition (and (actor ?c) (att ?l ?c) (att ?l ?c2) (evil ?c2)
                      (not (fear ?c)) (not (dead ?c2)))
       :goal (and (fear ?c))
)

(:behavior runFromEviltoUnknow
       :parameters (?c ?c2 - character ?l - location)
       :precondition (and (actor ?c) (att ?l ?c) (fear ?c) (att ?l ?c2)
                      (evil ?c2) (not (dead ?c2)) (< (defender ?c)1))
       :goal (and (not (nearchr ?c ?c2)) (not (att ?l ?c)))
)

(:behavior runFromEviltoDefence
       :parameters (?c ?c2 ?c3 - character ?l ?l2 - location)
       :precondition (and (actor ?c) (att ?l ?c) (fear ?c) (att ?l ?c2)
                      (att ?l2 ?c3) (evil ?c2) (not (evil ?c3))
                      (strong ?c3) (not (dead ?c2)))
       :goal (and (not (nearchr ?c ?c2)) (att ?l2 ?c))
)

(:behavior notNearUnknow
       :parameters (?c ?c2 - character)
       :precondition (and (actor ?c) (nearchr ?c ?c2) (not (known ?c ?c2))
                      (not (dead ?c2)))
       :goal (and (not (nearchr ?c ?c2)))
)

(:behavior calm
       :parameters (?c ?c2 - character)
       :precondition (and (actor ?c) (fear ?c) (not (actor ?c2)) (evil ?c2)
                      (dead ?c2))
       :goal (and (not (fear ?c)))
)

(:behavior getOutEvilStomach
       :parameters (?c ?c2 - character ?l ?l2 - location)
       :precondition (and (actor ?c) (not (actor ?c2)) (att ?l2 ?c)
          (att ?l ?c2) (evil ?c2) (stomach ?c2 ?l2) (near ?l ?l2))
       :goal (and (not (att ?l2 ?c)))
)
```

Figura A.10: Descrição da personagem Chapeuzinho

```
Events:

(:event passKnowledge
       :parameters (?c ?c2 - character ?l - location)
       :precondition (and (att ?l ?c) (att ?l ?c2) (talked ?c ?c2) (evil ?c)
                      (= (victim ?c) 0))
       :effect (and (increase (victim ?c) 1))
)

addKnowledge ?c
character grandma

(:event desguiseasgrandma
       :parameters (?c ?c2 - character ?l - location)
       :precondition (and (stomach ?c ?l) (att ?l ?c2) (not (desguised ?c)))
       :effect (and (desguised ?c))
)

changeComponent
character ?c
Face
Value: Wolf
Appeal: 5

Mouth
Value: Big
Appeal: 5

Cloth
Value: Dress
Appeal: 8


(:event adddefender
       :parameters (?c ?c2 - character ?l - location)
       :precondition (and (att ?l ?c) (att ?l ?c2) (not (evil ?c))
               (not (evil ?c2)) (strong ?c2) (= (defender ?c) 0))
       :effect (and (increase (defender ?c) 1))
)

(:event died
       :parameters (?c - character)
       :precondition (and (< (lifepoints ?c) 1) (not (dead ?c)))
       :effect (and (dead ?c))
)

changeComponent
character ?c
Cloth
Value: Pierced
Appeal: 7

Mouth
Value: Cut
Appeal: 7
```

Figura A.11: Descrição dos eventos