UNIVERSIDADE FEDERAL DE UBERLÂNDIA
FACULDADE DE CIÊNCIA DA COMPUTAÇÃO
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

MODELAGEM E ANÁLISE DE VIDEO GAMES USANDO AS WORKFLOW NETS E A LÓGICA LINEAR

GUILHERME WILLIAN DE OLIVEIRA

Uberlândia - Minas Gerais
2012

UNIVERSIDADE FEDERAL DE UBERLÂNDIA
FACULDADE DE CIÊNCIA DA COMPUTAÇÃO
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

GUILHERME WILLIAN DE OLIVEIRA

MODELAGEM E ANÁLISE DE VIDEO GAMES USANDO AS WORKFLOW NETS E A LÓGICA LINEAR

Dissertação de Mestrado apresentada à Faculdade de Ciência da Computação da Universidade Federal de Uberlândia, Minas Gerais, como parte dos requisitos exigidos para obtenção do título de Mestre em Ciência da Computação.

Área de concentração: Engenharia de Software.

Orientador:
Prof. Dr. Stéphane Julia

Uberlândia, Minas Gerais
2012

UNIVERSIDADE FEDERAL DE UBERLÂNDIA
FACULDADE DE CIÊNCIA DA COMPUTAÇÃO
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

Os abaixo assinados, por meio deste, certificam que leram e recomendam para a Faculdade de Ciência da Computação a aceitação da dissertação intitulada "**Modelagem e Análise de Video Games Usando as WorkFlow nets e a Lógica Linear**" por **Guilherme Willian de Oliveira** como parte dos requisitos exigidos para a obtenção do título de **Mestre em Ciência da Computação**.

Uberlândia, 29 de Fevereiro de 2012

Orientador: _____

Prof. Dr. Stéphane Julia
Universidade Federal de Uberlândia

Banca Examinadora:

_____

Prof. Dr. Jose Reinaldo Silva
Universidade de São Paulo, Escola Politécnica

_____

Prof. Dr. Michel dos Santos Soares
Universidade Federal de Uberlândia

UNIVERSIDADE FEDERAL DE UBERLÂNDIA
FACULDADE DE CIÊNCIA DA COMPUTAÇÃO
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

Data: Fevereiro de 2012

Autor:	**Guilherme Willian de Oliveira**
Título:	**Modelagem e Análise de Video Games Usando as WorkFlow nets e a Lógica Linear**
Faculdade:	**Faculdade de Ciência da Computação**
Grau:	**Mestrado**

Fica garantido à Universidade Federal de Uberlândia o direito de circulação e impressão de cópias deste documento para propósitos exclusivamente acadêmicos, desde que o autor seja devidamente informado.

_____

Autor

O AUTOR RESERVA PARA SI QUALQUER OUTRO DIREITO DE PUBLICAÇÃO DESTE DOCUMENTO, NÃO PODENDO O MESMO SER IMPRESSO OU REPRODUZIDO, SEJA NA TOTALIDADE OU EM PARTES, SEM A PERMISSÃO ESCRITA DO AUTOR.

©Todos os direitos reservados a Guilherme Willian de Oliveira

Dedicatória

Aos meus pais Afonso e Marli e aos meus irmãos.

Agradecimentos

Agradeço...
A todos que vem me ajudando a melhor a cada dia.

"O homem não é nada além daquilo que a educação faz dele."
(Immanuel Kant)

Resumo

O objetivo deste trabalho é apresentar uma abordagem baseada nas *WorkFlow nets* e na Lógica Linear para o processo de criação de video games. A ideia principal consiste em representar os cenários existentes numa *quest* por meio de um tipo particular de redes de Petri chamadas *WorkFlow nets*.

Um tipo de análise qualitativa com base nas árvores de prova da lógica linear pode então ser realizada a fim de provar a propriedade *"soundness"* que corresponde a uma *quest* consistente do ponto de vista do jogo.

A análise quantitativa preocupa-se com o planejamento do tempo de jogabilidade de uma *quest*. Ela se basea numa versão t-temporal das *WorkFlow nets* e no cálculo de datas simbólicas derivadas do cálculo dos sequentes da lógica linear.

Uma versão estendida das *WorkFlow nets* que autoriza a inclusão de recursos discretos é considerada. Tais recursos representam os diversos itens que o jogador pode encontrar durante as missões do jogo.

Em particular, é mostrado o efeito de se usar tais recursos no modelo de rede de *quest* que representa a integralidade dos cenários de *quests* do jogo.

Partes do jogo *The Legend of Zelda: Oracle of Ages* são usadas a fim de ilustrar a abordagem proposta.

Palavras chave: rede de petri, lógica linear, workflow net, soudness, video games.

Abstract

The objective of this work is to present an approach based on WorkFlow net and Linear Logic for the design process of video games. The main idea consists of representing the scenarios existing at a quest level by a particular type of Petri net called WorkFlow net.

A kind of qualitative analysis based on the proof trees of linear logic can then be performed in order to prove the correctness of the soundness property which corresponds to a consistent quest from the point of view of the game.

The quantitative analysis is concerned with the planning of the gameplay time of a quest. It is base on version of t-time WorkFlow nets and symbolic date calculation derived from the sequent calculus of linear logic.

An extended version of the WorkFlow nets which allows the inclusion of discrete resources permits representing in a formal way, the different items that the player can find and use during the quests of the game.

Parts of the game The Legend of Zelda: Oracle of Ages are used to illustrate the proposed approach.

Keywords: petri net, linear logic, workflow net, soundness, video games.

Sumário

Lista de Figuras

Lista de Tabelas

Capítulo 1

Introdução

A criação de jogos de *video game* envolve várias atividades específicas (como o *game design* e o *level design*) que não necessariamente existem quando se consideram processos de desenvolvimento de *software* mais tradicionais. Um dos propósitos principais do *game design* e *level design* é especificar as diferentes *quests* do jogo e os cenários que o jogador será autorizado a seguir a fim de alcançar alguns objetivos específicos. Geralmente, o *game design* depende de documentos textuais que ilustram o conceito do jogo e representam o documento de referência para todos os membros da equipe envolvida no processo de desenvolvimento do jogo. É na especificação do *level design* que os objetos disponíveis e as principais ações dos jogadores são definidos.

Pelo fato de os jogos terem-se tornado cada vez mais complexos, novos conceitos e ferramentas de desenvolvimento precisam ser encontrados.

De acordo com [Bruegge e Dutoit 2009], a engenharia de software pode ser vista essencialmente como uma atividade de modelagem. É então natural aplicar técnicas de modelagem no contexto do desenvolvimento de jogos. Em [Onder 2002] e [Lewinski 1999], um tipo de fluxograma chamado *story-board* é usado para mostrar as sequências de atividades que um jogador terá que seguir para cumprir determinados objetivos dentro de missões específicas ou níveis. Tal abordagem parece ser um primeiro passo para melhorar um documento de especificação textual simples mas representa apenas um tipo de representação visual dos cenários de jogo. Lamothe em [LAMOTHE 2002] utiliza gráficos de estado/transição para especificar o comportamento interno de jogos de *video game* a fim de ajudar à implementação de jogos baseados em técnicas de inteligência artificial.

Vários trabalhos, tais como os de Siang e Rao [SIANG e RAO 2004], Penton [Penton 2003] e Rucker [Rucker 2003] utilizam diagramas UML para mostrar como os diferentes objetos do jogo irão interagir de acordo com algumas ações que serão executadas pelo jogador. Em particular diagramas UML são interessantes para produzir uma estrutura de execução do jogo mas não mostram geralmente de forma explícita os possíveis cenários existentes dentro de uma missão ou de um nível do jogo.

Modelos formais têm sido usados em diferentes tipos de jogos de tabuleiro clássicos

como o xadrez. Por exemplo, Kaiser em [Kaiser 2007] define uma linguagem formal chamada de GDL (*game description language*). Tais linguagens de especificação formal são normalmente muito eficiente para especificar a lógica de funcionamento de certos jogos deterministas, mas não oferecem uma boa representação visual de cenários dos jogos.

Em [Natkin et al. 2004], um novo tipo de modelo de rede Petri, chamado redes de transação (*transaction nets*) foi definido para representar os cenários de missões em jogos de vídeo game. Em tal modelo, o jogador é o principal recurso do jogo e as ações são associadas aos lugares. Uma vez que o modelo de uma *quest* é produzido, alguns tipos de análise podem ser realizadas por meio de mecanismos de simulação baseados nos algoritmos clássicos de jogos de fichas, que mostram para cenários específicos como as fichas podem ser produzidas ou consumidas.

Nesta dissertação é proposta uma nova abordagem baseada nas *WorkFlow nets* para especificar os cenários existentes em um nível de *quest* de um video game. A lógica linear será usada em particular para provar o critério de corretude *soundness* mediante um tipo de análise qualitativa do jogo. Também será realizada uma análise qualitativa para o cálculo dos tempos de jogo das *quests* ainda na fase de *game design* (quando o jogo está sendo planejado e não existe ainda).

O Capítulo 2 apresenta os jogos de *video game*, sendo que a Seção 2.1 faz uma contextualização histórica, mostrando o surgimento dos jogos e sua importância no mercado de entretenimento. A Seção 2.2 descreve as principais técnicas de programação utilizadas na criação dos jogos de *video games*. A Seção 2.3 apresenta os principais modelos utilizados na criação de *video games* assim como as tentativas de formalização do processo.

O Capítulo 3 apresenta as redes de Petri e as *WorkFlow nets*, a Seção 3.1 faz uma breve introdução sobre as redes de Petri. A Seção 3.2 define as *WorkFlow nets* apresentando em particular o critério de corretude *soundness* que as caracteriza. Finalmente, a Seção 3.3 apresenta a lógica linear e suas relações com as redes de Petri.

O Capítulo 4 define o modelo desenvolvido para formalizar os cenários de *video games*. A Seção 4.1 descreve a estrutura clássica de um jogo de *video game*. A Seção 4.2 define cada parte do modelo para a construção de cenários de *quests*.

No Capítulo 5 são apresentadas abordagens para análise qualitativa e quantitativa das *WorkFlow nets*. A análise qualitativa é apresentada na Seção 5.1, a Seção 5.1.4 complementa a análise qualitativa, tratando os cenários que possuem recursos. Já a Seção 5.2 apresenta a análise quantitativa que trata em particular da técnica de cálculo dos tempos de jogos das *quests*.

Finalmente, o Capítulo 6 apresenta a conclusão deste trabalho e as perspectivas de trabalhos futuros.

Capítulo 2

Jogos Eletrônicos e Engenharia de Software

2.1 História dos Jogos Eletrônicos

A história dos *video games* envolve o desenvolvimento dos jogos eletrônicos. Começado na década de 1970, inicialmente em computadores tipo *mainframe*, logo culminou na criação de arcades e consoles de *video games*. Antes de os arcades apresentarem os *video games* para o público, eles eram disponíveis apenas para aqueles que tivessem acesso aos laboratórios de computação em universidades ou empresas. Iniciando com *PONG* em 1972, os *video games* obtiveram grande sucesso nos arcades. Consoles domésticos foram popularizados com a chegada do *Atari VCS*, posteriormente nomeado de *Atari 2600*, pelo qual foram produzidos jogos consagrados como *Space Invaders*, *Pac-Man* e *Defender* [Wolf 2007].

Nos anos 1970 surgiu um novo setor de mercado criado pelos jogos eletrônicos, com milhões de potenciais jovens consumidores. Porém apenas poucos desenvolvedores eram aptos a criá-los; não existia na época absolutamente nenhum livro sobre o tema. Com a chegada dos anos 1980 vieram os primeiros computadores de 16-bits como IBM PC, Mac, Amiga 500 e Atari ST, foi quando os jogos começaram a ter uma boa aparência, melhorando tanto em gráficos quanto em jogabilidade. Alguns jogos 3D, como *Wing Commander* e *Flight Simulator* foram lançados no mercado. Na mesma época os PCs foram lentamente ganhando popularidade pelos seus baixos preços e funcionalidades interessantes para o setor comercial. No início dos anos 1990, IBM PC e compatíveis eram os líderes dos computadores domésticos, mas não forneciam suporte suficiente para criar jogos atraentes como os desenvolvidos para consoles de *video game*. Em 1993, o jogo *DOOM* foi lançado na plataforma PC/DOS e tornou-se uma referência dos jogos, pois trouxe relevância à Microsoft e à plataforma PC. A capacidade de trabalho com vídeo e áudio de tempo real do Windows eram limitados. A Microsoft introduziu o chamado

Win-G para resolver os problemas com vídeo. Com adição de novas bibliotecas gráficas, de som, entrada, rede e sistemas 3D, nasceu o *DirectX* [LAMOTHE 2002].

Atualmente a indústria ligada aos jogos eletrônicos e aos videogames já é considerada como a maior indústria de entretenimento no mundo, ultrapassando o faturamento da indústria do cinema [V. Gal et al. 2002]. O mercado está dividido entre PC e consoles de jogos (PS2, PS3, Xbox, Wii, etc) com dois casos especiais: o jogo para o consoles portáteis (Game Boy, Palm, Celulares, PSP) e jogos totalmente Online (*Everquest*, *Tibia*, etc) [Gal et al. 2002].

Há numerosas classificações possíveis para *video games*. Neste trabalho vamos utilizar a definição de Rollins [Rollins e Morris 2004]. Essa classificação se baseia no foco principal do jogo:

- **Jogos de Ação:** Levar o jogador para pressionar o mais rápido possível uma sequência de botões. Jogos de Luta (*Beat them all*, *Street Fighter*) são um bom exemplo deste gênero.
- **Jogos de Aventura:** São, provavelmente os que estão mais relacionados ao cenário audiovisual clássico. O jogador é o herói de um cenário complexo. *Metal Gear Solid 2* é um jogo de aventura típico.
- **Jogos de estratégia:** Tem como principais características a complexidade das decisões tomadas pelo jogador num universo político, fantástico, econômico ou militar. *The Sims* e *Black and White* são jogos de estratégia bem elaborados.
- **Jogos de simulação:** Levam o jogador a exercitar um esporte simulado ou outro dispositivo físico (avião, carro, skate, etc).
- **Jogos de quebra-cabeça:** Tem como objetivo a resolução de um desafio analítico. As versões para computador de jogos clássicos (como o xadrez) são a base dos jogos de *Puzzle*. Os jogos de tipo *Puzzle* levam o jogador a ser encarregado de uma investigação. Jogos de mistério são um exemplo do gênero.
- **Jogos de descoberta:** Tem como objetivo principal descobrir uma história (um cenário desconhecido no começo do jogo) e resolver enigmas incorporado no jogo. Por exemplo, o jogo de *Versalhes* leva o leitor a descobrir a vida do rei Louis XIV por meio de um jogo de quebra-cabeça.

De acordo com Rollins, o que define o estilo é a forma como o jogo é executado. *Duke Nukem* e *Tomb Raider* são jogos de ação / aventura, mas seu estilo de fazer o jogador pensar pertence a um gênero diferente. RPG (*Role Playing Games*) constituem em uma mistura de ação e aventura com jogos de estratégia, como por exemplo *Diablo II*.

2.2 Paradigmas de Programação e de Desenvolvimento

Uma vez que o projeto inicial do jogo foi definido, a linguagem de desenvolvimento deve ser decidida. A escolha depende de muitos fatores, tais como a familiaridade da equipe com a linguagem de programação, as plataformas-alvo (como PlayStation ou o Microsoft Windows), os requisitos de velocidade de execução e a linguagem dos vários motores de jogo.

Atualmente, linguagens orientada a objetos e que compilem em binário (linguagem nativa da plataforma-alvo), como por exemplo a linguagem C++ são mais populares para o desenvolvimento de jogos. A linguagem assembly é necessária para programação de consoles de *video game*, em rotinas que precisam ser as mais rápidas possíveis, ou exigem sobrecarga limitada. Linguagens periféricas como C#, Python e Ada tiveram pouco impacto sobre a indústria e são usados principalmente por pessoas familiarizadas com as linguagens, embora C# seja muito popular para a criação de ferramentas para desenvolvimento de jogos [DAVISON 2005, PENTON 2005].

Linguagens de script de alto nível são cada vez mais utilizadas como extensões incorporadas ao jogo fundamental escrito em uma linguagem de programação de baixo ou médio nível, como C e C++ respectivamente. Muitos desenvolvedores criaram linguagens personalizadas para seus jogos, como *id Software's QuakeC* e *Epic Games' UnrealScript*. Outros optaram por linguagens existentes como Lua e Python num contexto de reutilizabilidade. Jogos de computadores moderno envolvem o desenvolvimento de software de grande porte (sistemas que podem ser compostos de milhões de linhas de código) [BETHKE 2003].

2.2.1 APIs e Bibliotecas

Uma decisão-chave em programação de jogos são quais APIs (*Application Programming Interface*) e bibliotecas usar. Existem inúmeras bibliotecas disponíveis que cuidam de tarefas fundamentais da programação de jogos. Algumas bibliotecas podem lidar com processamento de som ou de gráficos. Outras podem até mesmo lidar com tarefas de inteligência artificial (IA), tais como *pathfinding*. Há até mesmo motores de jogos inteiros que lidam com a maioria das tarefas de programação de jogos, apenas necessitando ser codificada a lógica do jogo.

Quais APIs e bibliotecas devem ser escolhidas depende muito da plataforma-alvo. Por exemplo, bibliotecas de desenvolvimento para o PlayStation 2 não estão disponíveis para o Microsoft Windows e vice-versa. No entanto, existem *frameworks* disponíveis que permitem ou facilitam o desenvolvimento multi plataforma. Assim os programadores podem criar um jogo em uma única linguagem e ter o jogo disponivel em diversas plataformas, como o Wii, PlayStation 3, Xbox 360, Xbox, PSP e Microsoft Windows.

Os gráficos são uma característica-chave que define a maioria dos jogos. Apesar de gráficos 2D serem usados para a maioria dos jogos lançados até meados da década de

1990, quase todos os jogos agora possuem gráficos 3D.

O sistema operacional mais popular para computador pessoal é o Microsoft Windows, visto que vem pré-instalado em quase noventa por cento dos PCs vendidos, possui uma base de usuários extremamente grande. As duas APIs gráficas 3D mais populares para o Microsoft Windows são Direct3D e OpenGL. DirectX é uma coleção de APIs para jogos. Direct3D é a API 3D do DirectX. Direct3D é disponibilizada livremente pela Microsoft, assim como o restante das APIs do DirectX. A Microsoft desenvolveu o DirectX para programadores de jogos e continua a adicionar recursos para a API. A especificação do DirectX não é controlada por uma comissão de arbitragem aberta e a Microsoft está livre para adicionar, remover ou alterar os recursos que ela disponibiliza. O Direct3D não é portável, é projetado especificamente para o Microsoft Windows e nenhuma outra plataforma (embora uma forma de Direct3D é usada no Xbox da Microsoft).

OpenGL é uma especificação API portável. Código escrito com OpenGL é facilmente portado entre plataformas distintas. Por exemplo, Quake II foi portado do Windows para o Linux por um fã do jogo. OpenGL é um padrão mantido pelo *OpenGL Architecture Review Board* (ARB). A ARB se reúne periodicamente para atualizar a norma, adicionando suporte para funcionalidades emergentes providas pelos hardware 3D mais recentes. Uma vez que é baseada em padrões e tem sido desenvolvida há mais tempo, OpenGL é geralmente usada para aprendizagem nas faculdades e universidades. Além disso, as ferramentas de desenvolvimento fornecidas pelos fabricantes de alguns consoles de *video games* (como o Nintendo GameCube, Nintendo DS e do PSP) usam APIs gráficas que se assemelham a OpenGL. OpenGL muitas vezes fica para trás nas atualizações de recursos devido à falta de uma equipe permanente do desenvolvimento e da exigência que as implementações apenas iniciem o desenvolvimento após a norma ter sido publicada. Os programadores que optam por usá-la podem acessar alguns últimos recursos de *hardware* 3D por meio de extensões que não são padronizados. A situação pode mudar no futuro já que o OpenGL *architecture review board* (ARB) passou o controle da especificação para o *Khronos Group*, em uma tentativa de resolver o problema.

2.2.2 Estrutura de um jogo

Desenvolver jogos difere muito em relação ao desenvolvimento dos demais tipos de *softwares*. *Video games* são *softwares* extremamente complexos, pois devem passar a sensação de tempo real com simulações físicas convincentes. A Figura 2.1 mostra o *loop* padrão dos *video games*, que basicamente recebe entradas, avalia uma certa lógica e desenha imagens na tela [LAMOTHE 2002, LLOPIS 2003].

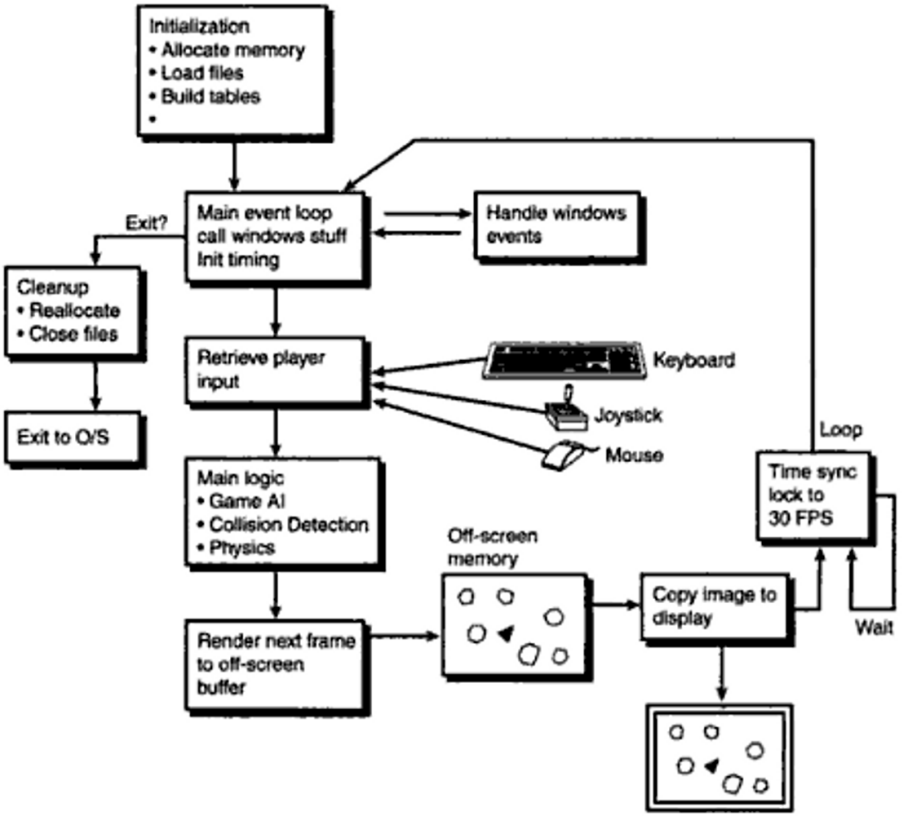

Figura 2.1: Arquitetura geral de loop dos jogos

2.3 Modelos para o desenvolvimento de jogos eletrônicos

Onder em [Onder 2002] descreve o uso de diagramas tipo *story-beat* mostrado na Figura 2.2 como exemplo. Um *story-beat* é constituído por uma coleção de formas ovais e de flechas a fim de mostrar o fluxo (ou fluxo alternativo) de uma história de jogos de computador. Diagramas *story-beat* mostram em alto nível como o jogador pode mover-se por meio de um jogo.

Para cada cena no diagrama *story-beat* tem-se a necessidade de gravar a localização, descrição, os objetos e membros do elenco em cena, assim como uma tabela com os eventos que descrevem os acontecimentos em resposta a cada jogador que pertence à cena descrita [Onder 2002].

Rollings e Morris em [Rollins e Morris 2004] sugerem o uso de matrizes de interação de *token* para a concepção de jogos de computador. Um exemplo é mostrado na Figura 2.3. Uma matriz de interação de *token* é uma tabela com todas as interações que ocorrem em um jogo de computador ou de um segmento de um jogo de computador. Um token é definido como um elemento do jogo discreto que será diretamente ou indiretamente manipulado pelo jogo.

Lewinski em [Lewinski 1999] descreve o uso de fluxogramas para o *design* de alto

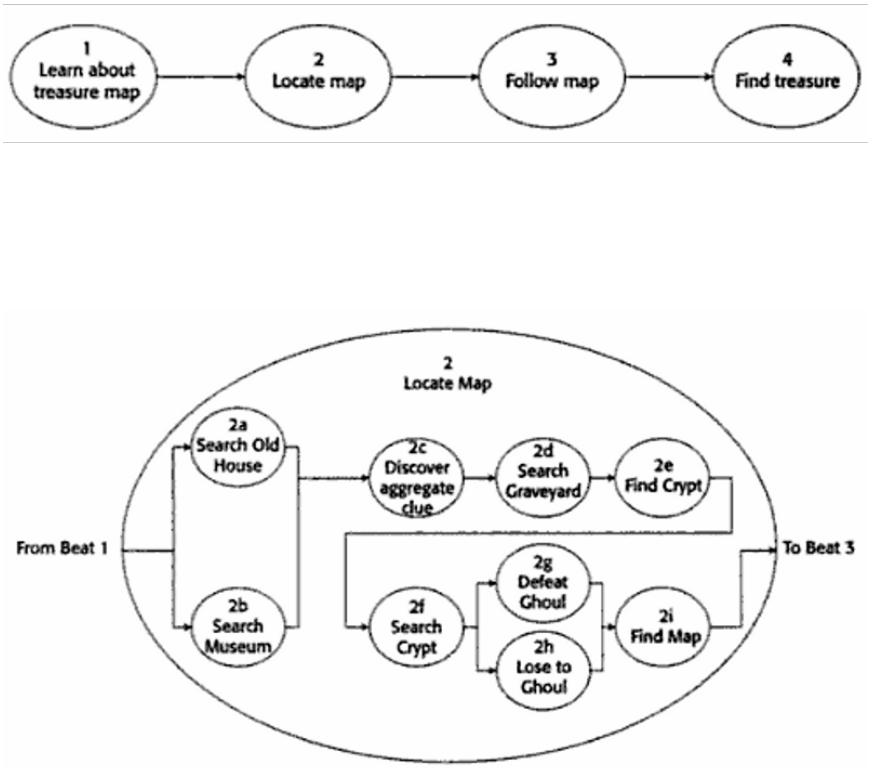

Figura 2.2: Digramas *story-beat*

nível dos jogos de computador. Os fluxogramas simplesmente mostram como cada cena ou missão em um jogo de computador (representado por uma elipse) flui para a próxima cena ou missão por meio de uma seta.

Rouse em [Rouse 2004], sugere o uso de *storyboards* (os mesmos usados em produção cinematográfica e televisiva) como mostrado na Figura 2.4. Eles permitem apresentar como o jogo irá aparecer para o leitor. Também, incentiva a produção de documentos de projeto técnico tais como a estrutura geral do código e as principais classes e pseudocódigo para projetos mais detalhados.

LaMothe em [LAMOTHE 2002] defende o uso dos diagramas de transição/estados a fim de apoiar as atividades de programação do jogo como mostrado na Figura 2.5.

Siang e Rao em [SIANG e RAO 2004] e Bethke em [BETHKE 2003] descrevem uma abordagem para a concepção de jogos de computador usando os diagramas de casos de uso e diagramas de classe da UML (*Unified Modeling Language*). Penton em [Penton 2003] afirma que uma das primeiras tarefas a fazer quando se projeta um jogo de computador é estabelecer as principais classes que serão utilizadas no jogo.

Rucker em [Rucker 2003] propôs a utilização de um maior subconjunto de UML para o desenvolvimento de jogos de computador, incluindo os diagramas de componentes para

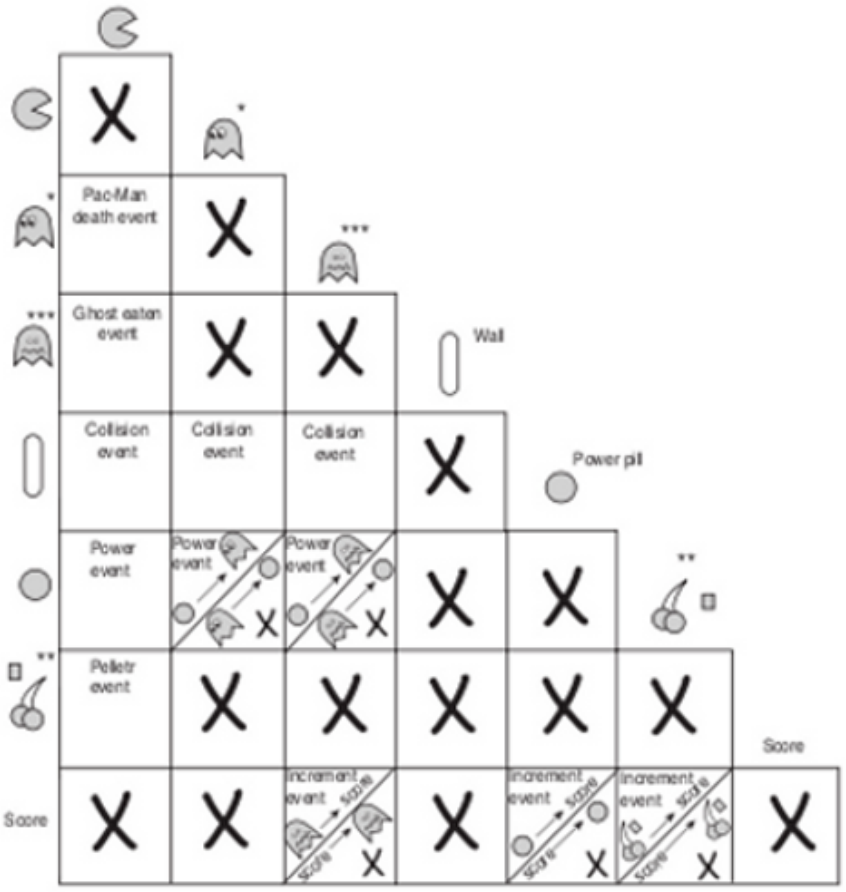

Figura 2.3: Matriz de interação de token do jogo *pacman*

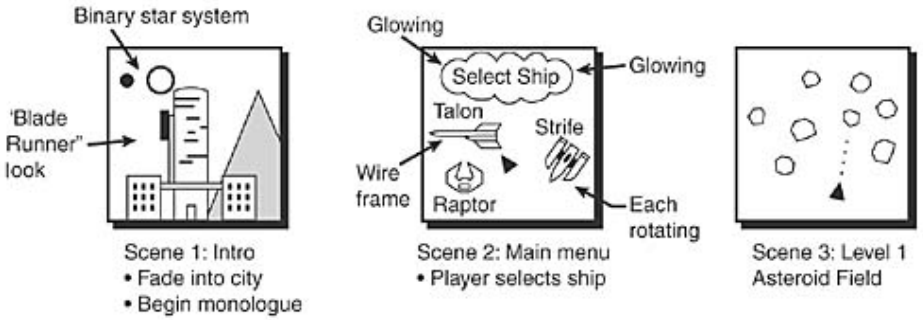

Figura 2.4: storyboard

as dependências dos arquivos de código fonte, diagramas de atividade para o fluxo de execução do programa, diagramas de sequência para a interação de objetos do programa, além dos diagramas de caso de uso e de classe. As abordagens descritas por Siang e Rao [SIANG e RAO 2004], Bethke [BETHKE 2003] e Rucker [Rucker 2003] mostram como os diagramas de caso de uso apresentam claramente as ações de alto nível que o jogador pode realizar, como é mostrado na Figura 2.6 por exemplo. Da mesma forma, os diagramas de classe são usados em alto nível para descrever como as ações se relacionam

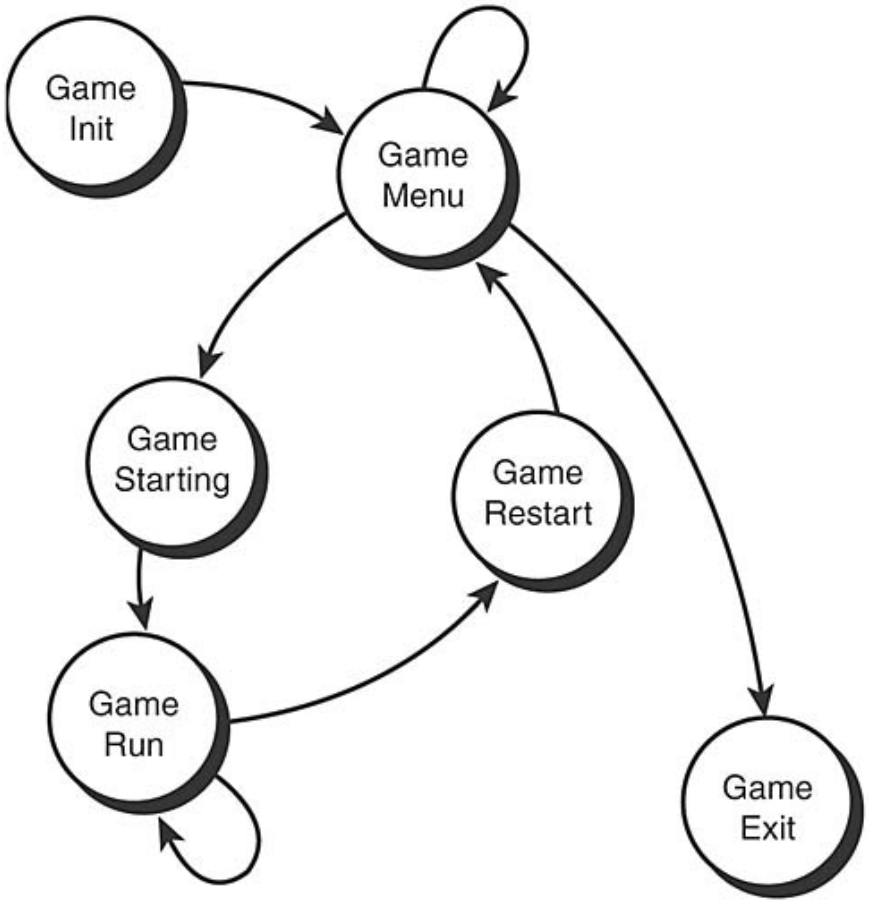

Figura 2.5: Diagrama de transição de estados para desenvolvimento de jogo

entre si. As abordagens descritas por eles geralmente não exemplificam os modelos de fluxos de níveis específicos dos jogos.

Gold em [Gold 2004] usa os gráficos para especificar os roteiros dos níveis de um jogo. Os vértices representam cenas do jogo e as arestas as transições entre as cenas. Um exemplo é apresentando na Figura 2.7.

2.3.1 Métodos formais para desenvolvimento de *video games*

Métodos formais são linguagem/métodos matemáticos, muitas vezes apoiadas por ferramentas, para desenvolver *software* e *hardware*. O rigor matemático permite aos utilizadores analisar e verificar esses modelos em qualquer parte do ciclo de vida do projeto de *software*: engenharia de requisitos, especificação, arquitetura, design, implementação, teste, manutenção e evolução [Woodcock et al. 2009].

O primeiro passo vital em um processo de alta qualidade de desenvolvimento de software é a engenharia de requisitos. Os métodos formais podem ser úteis para articular e representar requisitos [George e Vaughn 2003]. Suas ferramentas podem oferecer suporte automatizado necessário para verificar integridade, rastreabilidade, verificabilidade, reuti-

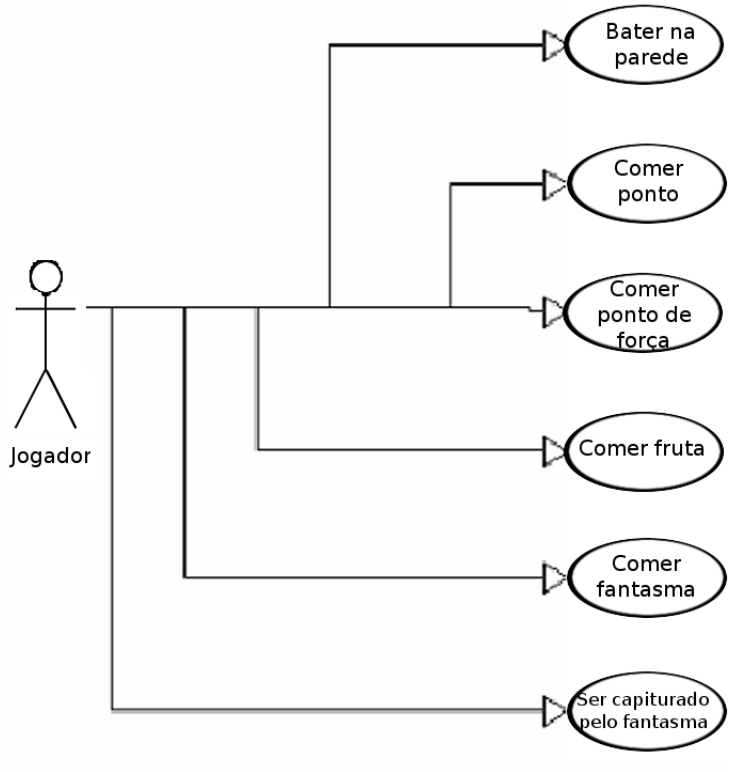

Figura 2.6: Diagramas de caso de uso

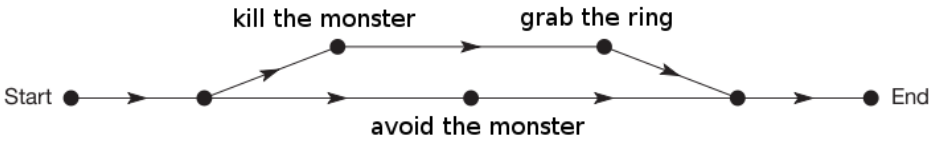

Figura 2.7: mapa lógico simples

lização, gerenciamento de inconsistência e para apoiar a evolução dos requisitos [Ghose 2000].

Os métodos formais podem ser usados na especificação de *software*. Assim fornecendo aos desenvolvedores uma declaração precisa do que o *software* irá realizar. Exemplos destes métodos incluem ASM (*Abstract State Machines*) [Abrial 1996, Borger e Stark 2003] e VDM (*Vienna Development Method*) [Jones 1990]. A especificação é um contrato entre o cliente e o programador, a fim de fornecer para ambos uma compreensão comum do objetivo do *software*. O cliente usa a especificação para orientar a aplicação do *software* e o programador usa a especificação para guiar a sua construção. Uma especificação complexa pode ser decomposta em subespecificações, cada uma descrevendo um subcomponente do sistema que pode ser delegada a outros programadores, sendo que um programador em um nível superior se torna um cliente de outro (*design by contract*) [Meyer 1991].

Sistemas de *software* complexos, como por exemplo os jogos, exigem cuidadosa organização da estrutura de seus componentes: um modelo de sistema elimina detalhes de implementação, permitindo que o arquiteto se concentre nas análises e decisões que são

cruciais para a estruturação do sistema a fim de satisfazer seus requisitos [Allen e Garlan 1992, van Lamsweerde 2003]. Por exemplo, *Wright* [Allen e Garlan 1996] é um exemplo de uma linguagem de descrição arquitetural baseada na formalização do comportamento de componentes de arquiteturas abstratas.

Os métodos formais podem ser usados no *design* de *software* para refinamento de dados que envolvem: especificação da máquina de estados, funções de abstração e provas de simulação. Hoare em [Hoare 2002] focaliza em métodos como VDM [Jones 1990] e cálculos de refinamento do programa [Dijkstra 1975, Morris 1987, Morgan 1988, Back e von Wright 1990].

Na implementação, métodos formais são usados para verificação do código. O método das asserções indutivas por exemplo foi inventado por Floyd e Hoare [Floyd 1967, Hoare 1969] e envolve notações matemáticas que são as relações mantidas entre as variáveis do programa e os valores de entrada iniciais. O código também pode ser gerado automaticamente a partir de modelos formais como em [Woodcock et al. 2009].

Kaiser demonstra em [Kaiser 2007] a utilização da linguagem GDL (*Game Description Language*) em conjunto com o Stanford GGP framework (*Stanford General Game Playing framework*). GDL é uma linguagem formal para a definição de jogos *determinísticos* com *informação perfeita*. Um jogo é dito *jogo de informação perfeita* se todos os jogadores possuem informação completa do estado atual. *Othello* é um jogo de informação perfeita porque todos estados são completamente capturados pela posição das peças no tabuleiro. Jogos em que os agentes não têm acesso a todos os estados,tais como o *Poker* ou *Scrabble*, não são jogos de informação perfeita. Um jogo determinístico é aquele no qual o estado atual é decidido inteiramente pelas decisões combinadas dos concorrentes. *Damas* é um jogo determinístico, mas os jogos envolvendo o rolar de dados ou cartões de embaralhar são considerados não-determinísticos. O Stanford GGP *framework* define como agentes participantes competem. É dada a descrição do jogo ao agentes GGP, seu papel dentro do jogo, limite de tempo disponível para analisar o jogo e o tempo disponível para apresentar lances. Cada jogador se comunica com um mediador central, o *Gamemaster* através de uma conexão *HTTP*. O *Gamemaster* transmite todas as informações do jogo para os jogadores, incluindo a descrição do jogo, hora de início, os movimentos do jogador e pontuação final do jogo. Jogadores se comunicam apenas com o *Gamemaster*, em particular enviando os lances legais em momento oportuno. Depois de cada turno, o *Gamemaster* informa os movimentos de cada jogador até uma posição terminal ser alcançada ocasionando o fim de jogo. Agentes GGP devem ser totalmente autônomos, ou seja, sem a intervenção de controle humano.

A linguagem GDL é uma variante da lógica de primeira ordem com base no KIF (*Knowledge Interchange Format*). Um jogo para demonstração é descrito na Figura 2.8. Cada agente GGP deve ser capaz de jogar qualquer jogo dada sua descrição. Neste exemplo, há apenas um jogador (`role white`), um estado do jogo inicial (`init (cell`

`1 1 b)`), e um movimento legal (`legal white (mark 1 1)`). O jogo acaba quando o jogador faz sua primeira jogada e só existe uma jogada legal. Usando um provador de teoremas, um agente GGP pode determinar todos os movimentos legais, condições de fim de jogo, valores da meta e calcular os sucessivos estados tendo como base o estado atual do jogo.

```
1.    (role white)
2.    (init (cell 1 1 b))
3.    (<= (legal white (mark ?x ?y))
          (true (cell ?x ?y b)))
4.    (<= (next (cell ?m ?n x))
          (does white (mark ?m ?n))
          (true (cell ?m ?n b)))
5.    (<= (goal white 100)
          (true (cell 1 1 x)))
6.    (<= (terminal)
          (true (cell 1 1 x)))
```

Figura 2.8: Jogo de demonstração em GDL

Este simples exemplo mostrado na Figura 2.8 contém todos os elementos-chave da descrição do jogo em GDL, incluindo as palavras-chave : `role, init, true, legal, does, goal, terminal`. Tokens, como `cell` e `mark` são específicos do jogo e não têm nenhum significado intrínseco. Os números são tratados como símbolos os quais não possuem nenhum significado fora do que foi definido na descrição do jogo em si.

2.3.2 Uso das Rede Petri no desenvolvimento de *video games*

A teoria das Redes de Petri foi proposta em 1962 por Carl Adam Petri, em sua tese de doutorado *Kommunikation mit Automaten*, na Universidade de Bonn, Alemanha.

As redes de Petri podem também ser vistas como uma ferramenta de modelagem gráfica e matemática aplicável a vários sistemas. Possuem características que as tornam promissoras para descrever e estudar sistemas de processamento de informação. Permitem representar formalmente características como a concorrência, o assincronismo, a distribuição, o paralelismo e o não determinismo. Como ferramenta gráfica, as redes de Petri podem ser usadas como um auxílio de comunicação visual semelhante aos fluxogramas e aos diagramas de blocos. Além disso, fichas são utilizadas para simular as atividades dinâmicas e concorrentes dos sistemas [Murata 1989].

O processo de criação de um jogo de computador do ponto de vista clássico da indústria é decomposto em duas fases: *Game Design* e *Level Design*. O *Game Design* é tido como

a fase que define a época, o estilo, o contexto, os objetivos a serem alcançados, principais tipos de objetos envolvidos, ou seja, a percepção do jogo pelos utilizadores. O *Level Design* é a fase de especificação que consiste em definir um espaço virtual, criar os quebra-cabeças, descrever as principais ações e conjunto de objetos utilizados para completar um dado objetivo.

Natkin e Vega [Natkin et al. 2004] foram os primeiros a utilizar as redes de Petri para modelar a ordenação de sequências de ação em um *video game*. A abordagem proposta permite descrever em particular a estrutura lógica das missões do jogo. A vantagem de usar redes de Petri (PN) é que elas podem ser representadas tanto por diagramas visuais quanto por modelos matemáticos, dependendo da complexidade do sistema estudado e do contexto de aplicação [Peterson 1981].

O modelo definido por Natkin e Vega é chamado de rede de transações. Duas operações **a** e **b**, de uma rede de transações podem ser combinadas como mostrado na Figura 2.9.

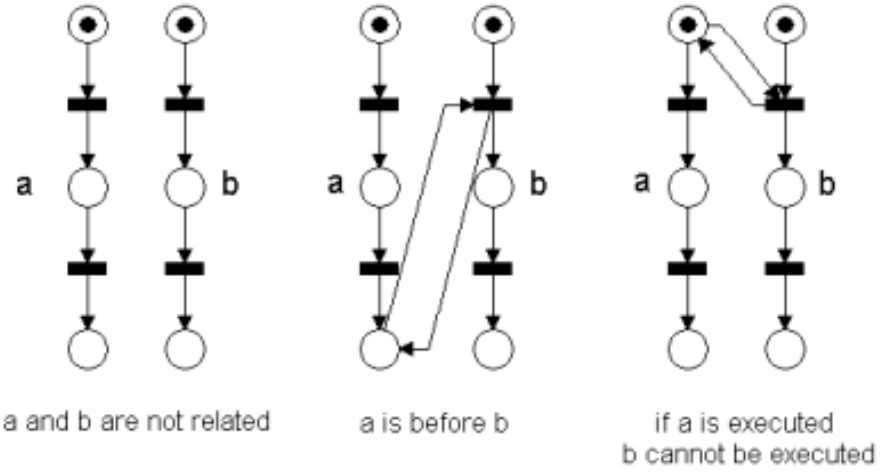

Figura 2.9: Relações entre duas operações a e b

Pela combinação de construções, podem ser geradas semânticas mais complexas. Por exemplo exclusão mútua entre duas operações, ou seja, se uma operação é executada então a outra não poderá sê-lo.

O primeiro exemplo apresentado por Natkin em [Natkin et al. 2004] é aplicado ao jogo *Silent Hill 2* (Konami 2002), um jogo de aventura e terror que ocorre em uma cidade misteriosa e quase deserta. O jogador controla um avatar (James), com objetivos claros desde o início. Para cumprir sua missão, ele deve explorar o ambiente, lutar contra os inimigos e coletar objetos (chaves, notas, informações) a fim de resolver os quebra-cabeças. O jogo começa em um estacionamento fora da cidade.

Para completar o primeiro nível o jogador precisa executar a sequência de ação principal mostrada na Figura 2.10:

(a) obter o mapa no carro de James e pegar a estrada para Silent Hill.

Uma vez na cidade,(c) deve vencer a luta contra uma criatura, a fim de poder continuar,

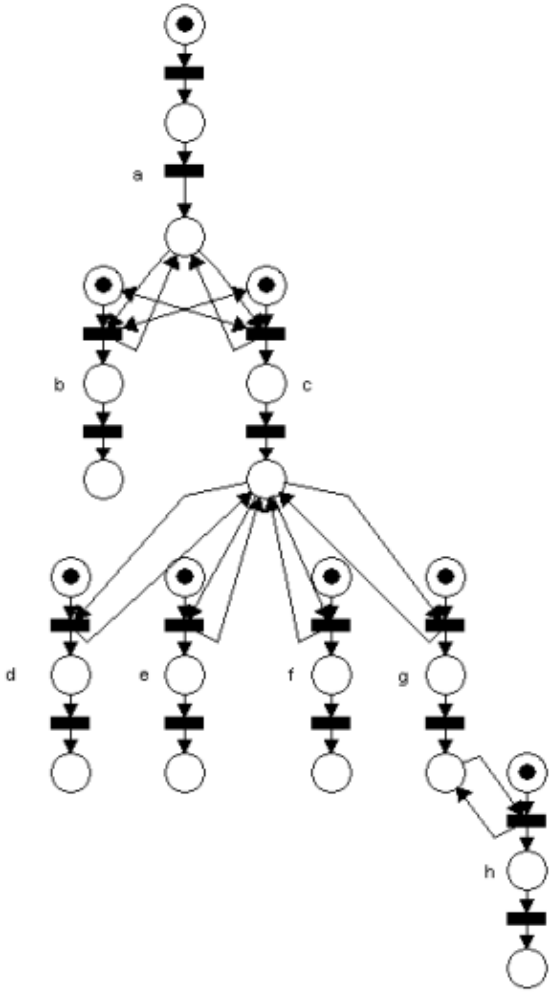

Figura 2.10: Silent Hill 2 Primeiro Nível

(b) Perder a luta leva ao termino do jogo.

Depois de vencer a luta, James pode executar as quatro próximas ações em ordem arbitrária:

(d) Recuperar uma inscrição obscura em um monumento estranho.

(e) Olhar o mapa no *trailer*.

(f) Encontrar um segundo mapa no bar do Neely.

(g) Encontrar a chave do apartamento em um corpo.

(d), (e) e (f) são ações opcionais que podem ser repetidas um número infinito de vezes, sem alterar o curso do jogo. (G) é obrigatório e marca o fim do primeiro nível. Depois disso, James é capaz de (h) entrar no apartamento para começar o próximo nível.

Capítulo 3

WorkFlow nets e Lógica Linear

3.1 Redes de Petri

As redes de Petri podem ser vistas como uma técnica formal de especificação de sistemas a eventos discretos bem estabelecida, que possibilita uma representação matemática e possui mecanismos de análise poderosos, permitindo a verificação de boas propriedades do sistema especificado. Usando-as redes de Petri, pode-se modelar sistemas com atividades paralelas, concorrentes, assíncronas e não-determinísticas [Valette 1980].

3.1.1 Definição de uma Rede de Petri Autônoma

As redes de Petri são um tipo de grafo composto por dois tipos de nós chamados lugares e transições. Um lugar pode ser representado por um círculo e uma transição por uma barra. Arcos conectam os lugares às transições ou as transições aos lugares, podendo ser rotulados por pesos (inteiros positivos) quando diferentes de 1. As fichas são utilizadas para simular o comportamento dinâmico dos sistemas.

Formalmente [Cardoso e Valette 1997] uma rede de Petri é uma quádrupla (P,T,Pré,Pós) onde:

- P é um conjunto finito de lugares;
- T é um conjunto finito de transições;
- Pré é uma relação que define os arcos que ligam os lugares às transições;
- Pós é uma relação que define os arcos que ligam as transições aos lugares;

Um lugar p é chamado de lugar de entrada de t se, e somente se, existe um arco direcionado de p para uma transição t. Por exemplo, considerando a rede de Petri da Figura 3.1(a), os lugares P1 e P2 são lugares de entrada da transição *t1*.

Um lugar p é chamado de lugar de saída de t se, e somente se, existe um arco direcionado de t para p. Por exemplo, considerando a rede de Petri da Figura 3.1(a), o lugar P3 é um lugar de saída da transição *t1*.

Um lugar p contém em um dado momento zero ou mais fichas (tokens) que são representadas por pontos pretos. Por exemplo, considerando a rede de Petri da Figura 3.1(a), o lugar P1 contém uma ficha e o lugar P2 está vazio, ou seja, não contém fichas. A marcação de uma rede de Petri refere-se à distribuição de fichas nos lugares, sendo que o número de fichas pode mudar durante a execução da rede. As transições são componentes ativos em uma rede de Petre, elas mudam a marcação da rede de acordo com as seguintes regras de disparo.

- Uma transição t é dita sensibilizada se, e somente se, cada lugar de entrada p de t contém pelo menos uma ficha. Por exemplo, a transição *t1* da rede de Petri da Figura 3.1(a) não está sensibilizada. Já a transição *t1* da rede de Petri da Figura 3.1(b) está sensibilizada, pois há uma ficha em cada lugar de entrada desta transição.
- Uma transição sensibilizada pode ser disparada. Se a transição t disparar, então t consome uma ficha de cada lugar de entrada p de t e produz uma ficha em cada lugar de saída p de t. As redes de Petri das Figuras 3.1(b) e 3.1(c) mostram um exemplo de disparo de transição. Neste caso, *t1* consome uma ficha de cada lugar de entrada (P1 e P2) e produz uma ficha em cada lugar de saída (P3).

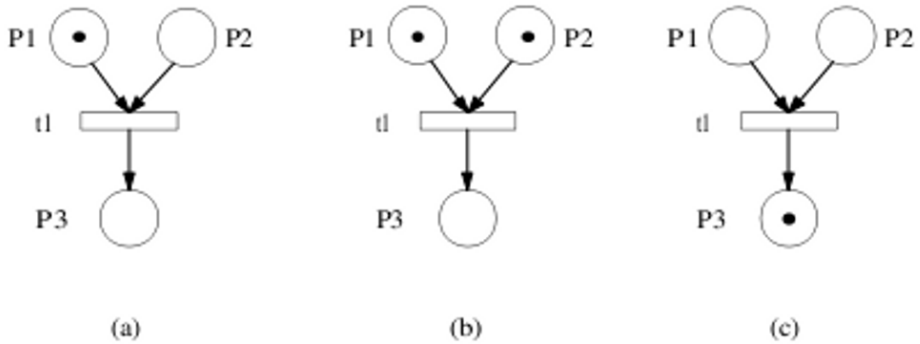

Figura 3.1: Exemplos de sensibilização e disparo de transição em uma rede de Petri

3.1.2 Definição das Redes de Petri Temporizadas

Esses modelos são empregados em sistemas em que o tempo é um fator importante. De um modo geral, as redes de Petri temporais são bastante adequadas para serem utilizadas em aplicações de simulação, diagnóstico e supervisão, análise de desempenho, mas também podem ser usadas para resolver o problema do escalonamento de sistema de produção [Julia 1998].

Nas redes de Petri p-temporizadas o tempo é representado por durações associadas aos lugares do modelo [Sifakis 1977, Tazza 1987] enquanto nas redes de Petri t-temporizadas, ele é representado por durações (números racionais positivos ou nulos) associadas às transições [Ramchandani 1974]. Nas redes de Petri t-temporais [Merlin 1974, Menasche 1982]

o tempo é representado por um intervalo $[\theta_{min}, \theta_{max}]$ associado a cada transição; a duração de sensibilização deve ser maior que o θ_{min} e menor que θ_{max}, ou seja, a transição só pode ser disparada dentro do intervalo de tempo $[\theta_{min}, \theta_{max}]$. Já nas redes de Petri p-temporais, o tempo é representado por um intervalo de tempo $[\theta_{min}, \theta_{max}]$ associado ao lugar p. θ_{min} representa a data mínima depois da qual uma ficha que está no lugar p torna-se disponível para o disparo de uma transição. Depois de θ_{max} a ficha torna-se "morta", o que caracteriza uma violação de restrição e um funcionamento anormal do sistema.

3.1.3 Definição de Redes de Petri de alto nível

As redes de Petri de alto nível, além de representarem o estado do sistema pela distribuição das fichas, têm a capacidade de carregar informações. Alguns tipos de redes de Petri de alto nível são apresentadas a seguir:

- **Redes de Petri Coloridas** [David e Alla 2004, Jensen 1991]: nessas redes cores são associadas às fichas com o objetivo de diferenciá-las, o que permite representar diferentes processos e recursos. Em [Moncelet et al. 1998], por exemplo, foi utilizado um modelo de rede de Petri Colorida para a modelagem de sistemas de produção híbridos;
- **Redes de Petri Predicado/Transição** [Cardoso et al. 1999, Genrich 1987]: descrevem de maneira estruturada o conjunto *controle e dados*. Em [Champagnat et al., 1998], por exemplo, foram integradas a uma sequencia de equações diferenciais e algébricas para a modelagem de uma estocagem de gás.
- **Redes de Petri a Objetos** [Silbertin 1985, Cardoso et al. 1999]: podem ser consideradas como uma extensão das Redes de Petri Predicado/Transição que aborda o conceito de paradigma orientado a objetos. A vantagem dessa modelagem é a possibilidade de se aproveitar da capacidade das redes de Petri para representar concorrência, controle de fluxo e restrições e ao mesmo tempo, beneficiar-se da modularidade da orientação a objetos. Em [Sanders 1998], foram usadas para a modelagem de sistemas de produção complexos com problemas baseados em restrições.
- **Redes de Petri Estocásticas** [Florin e Natkin 1984]: um tempo aleatório é associado ao disparo de uma transição. Esse modelo é usado para sistemas nos quais os eventos não podem ser bem definidos, como, por exemplo, o tempo entre a falha de uma máquina e outra. Para esse modelo o tempo é definido através de uma distribuição que segue uma lei exponencial, o que permite associar a rede a um processo Markoviano equivalente.

- **Redes de Petri Fuzzy** [Scarpelli e Gomide 1993, Cardoso et al. 1999]: é um formalismo que combina a teoria de conjunto *fuzzy* e a teoria de rede de Petri. Sendo assim, é uma ferramenta que representa a incerteza do conhecimento sobre um estado.

3.1.4 Definição das boas propriedades

Verificar as boas propriedades de um modelo é um fator crucial para que ele funcione de maneira satisfatória atendendo os requisitos do sistema. É possível, por meios analíticos, verificar as propriedades de uma rede de Petri. A seguir serão apresentadas algumas propriedades das redes de Petri.

Limitabilidade: *uma rede é limitada ou k-limitada se o número de fichas em cada lugar não excede um número finito k para qualquer marcação alcançável a partir da marcação inicial.* O algoritmo que permite decidir se uma rede de Petri marcada é k-limitada é baseado na construção de uma árvore chamada *árvore de cobertura* [Cardoso e Valette 1997]. Parte-se da marcação inicial e cada transição sensibilizada por essa marcação dá origem a um ramo. As marcações obtidas através do disparo das transições são calculadas e o processo recomeça para cada marcação obtida. A construção de um ramo é interrompida desde que seja encontrada uma marcação:

- igual a uma outra já calculada e para a qual todos os sucessores já foram ou serão calculados.
- estritamente superior a uma marcação do *ramo que está sendo explorado.*

No segundo caso, para-se a exploração da árvore, pois a rede marcada não é limitada. O valor $M'(p)$ que torna a marcação estritamente superior é notado w.

Vivacidade: *uma rede é considerada viva se toda transição t pode ser sensibilizada a partir de qualquer marcação M' do grafo de marcações alcançáveis. Esse conceito, na prática, garante que o sistema será livre de bloqueios (deadlock free).*

Se o grafo não for conexo (rede não reiniciável), a rede é viva se, em cada componente pendente fortemente conexo (ou subgrafo conexo), todas as transições $t \in T$ etiquetarem ao menos um arco [Cardoso e Valette 1997].

Blocos bem formados: Se o disparo de uma transição na rede de Petri é considerado não instantâneo, torna-se possível substituir uma transição numa rede $\mathcal{P}$ por outra rede $\mathcal{P}'$ chamada de bloco bem formado [Valette 1979]. Por exemplo, na Figura 3.2 a transição t_3 da rede de Petri da Figura 3.2 *a)* pode ser substituída pelo bloco bem formado representado pela rede de Petri da Figura 3.2 *b)*. O resultado é a rede de Petri da Figura 3.2 *c)*. Como a rede de Petri da Figura 3.2 *b)* é um bloco bem formado, de acordo com as definições apresentadas em [Valette 1979], A rede de Petri da Figura 3.2 *c)* continua com as boas propriedades da rede de Petri da Figura 3.2 *a)*.

Isso permite analisar a estrutura de controle por refinamentos gradativamente.

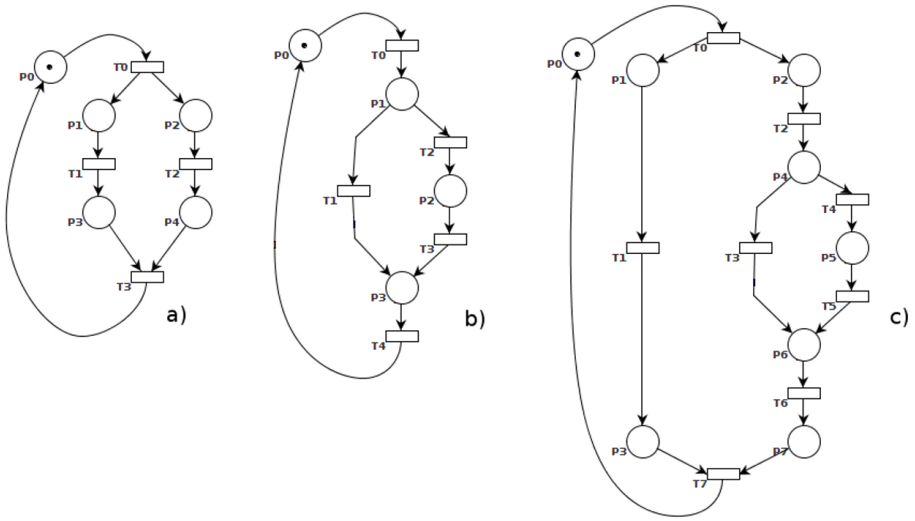

Figura 3.2: Exemplos de Bloco Bem Formado

3.2 WorkFlow Nets

3.2.1 Definição

Uma rede de Petri que modela um processo de *WorkFlow* é uma *WorkFlow net* [van der Aalst e van Hee 2004], [van der Aalst 1998]. Uma *WorkFlow net* satisfaz as seguintes propriedades [van der Aalst 1998]:

- Tem apenas um lugar de início (denominado *Start*) e apenas um lugar de término (denominado *End*), sendo esses dois lugares tratados como lugares especiais; o lugar Start tem apenas arcos de saída e o lugar End apenas arcos de entrada.
- Uma ficha em Start representa um caso que precisa ser tratado e uma ficha em End representa um caso que já foi tratado.
- Toda transição T e lugar P deve estar em um caminho que se encontra entre o lugar Start e o lugar End.

3.2.2 Processos

Um processo define quais tarefas precisam ser executadas e em que ordem a execução deve ocorrer. Modelar um processo de WorkFlow em termos de uma *WorkFlow net* é bem direto: transições são componentes ativos e modelam as tarefas, lugares são componentes passivos e modelam as condições (pré e pós) e as fichas modelam os casos [van der Aalst 1998].

Para ilustrar o mapeamento de processos em *WorkFlow nets*, considera-se o processo de tratamento de reclamações apresentado em [van der Aalst e van Hee 2004]: “uma reclamação é inicialmente gravada. Então, o cliente que efetuou a reclamação e o departamento

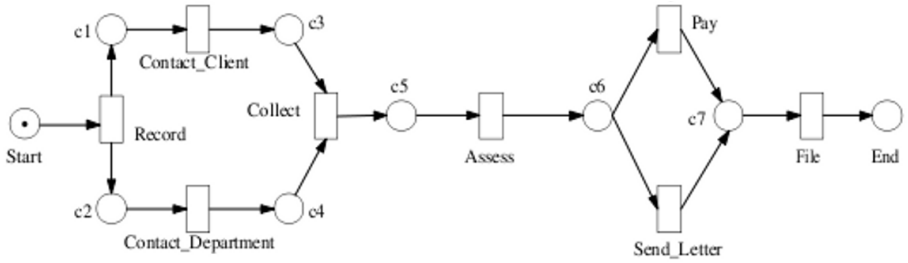

Figura 3.3: *WorkFlow net* para o processo de tratamento de reclamações.

responsável pela reclamação são contactados. O cliente é questionado para maiores informações. O departamento é informado sobre a reclamação. Estas duas tarefas podem ser executadas em paralelo, isto é, simultaneamente ou em qualquer ordem. Depois disso, os dados são recolhidos e uma decisão é tomada. Dependendo da decisão, ou um pagamento de compensação é efetuado, ou uma carta é enviada. Finalmente, a reclamação é armazenada". A Figura 3.3 mostra a *WorkFlow net* que representa esse processo.

3.2.3 Propriedades

Soundness é um critério de verificação de correção definido para as *WorkFlow nets*. Em particular uma *WorkFlow net* é *sound* se, e somente se, as três condições a seguir forem satisfeitas:

- Para cada ficha colocada no lugar de início, uma (e apenas uma) ficha aparecerá no lugar de término.
- Quando uma ficha aparece no lugar de término, todos os outros lugares estão vazios, considerando o caso em questão.
- Considerando uma tarefa associada à uma transição, é possível evoluir da marcação inicial até uma marcação que sensibiliza tal transição, ou seja, não deve haver nenhuma transição morta na *WorkFlow net*.

A propriedade *soundness* é um critério importante a ser satisfeito quando se trata de processos de *WorkFlow* e a prova desse critério está relacionada com a análise qualitativa, no contexto das *WorkFlow nets*.

Em [van der Aalst e van Hee 2004], são apresentados três métodos para provar a propriedade *soundness*: o primeiro método é baseado na construção de grafos de alcançabilidade, o segundo é baseado na análise das propriedades *liveness* e *boundedness* para uma rede *"short-circuited"* e o terceiro é baseado na substituição de blocos bem formados. Em [van der Aalst 1997], por exemplo, o método para provar *soundness* é baseado na construção de grafos de cobertura que podem consumir muito tempo em sua construção. O autor também propõe uma nova classe de *WorkFlow nets*, as *"Free-choice*

WorkFlow nets" que permitem determinar a propriedade *soundness* em tempo polinomial. Em [van der Aalst 1996] são apresentadas as *"Well-Structured WorkFlow nets"* que têm um número desejável de propriedades e para as quais a propriedade *soundness* também pode ser verificada em tempo polinomial. Em [Bi e Zhao 2004] é proposto um método baseado em lógica proposicional para verificar boas propriedades em processos de workflow. Em particular, um formalismo de modelagem de *WorkFlow*, baseado em atividades, para verificação baseada em lógica é apresentado. Nesse caso, a complexidade do algoritmo apresentado para a verificação é $O(n^2)$, mas esse algoritmo não é completo e não pode tratar alguns padrões de *WorkFlow*. Em [Li e Song 2005], um subconjunto das WorkFlow nets clássicas é proposto: as *"Normalized WorkFlow nets"*. As *"Normalized WorkFlow nets"* adicionam algumas restrições estruturais aos padrões de roteamento das *WorkFlow nets* e garantem a propriedade *soundness*. Entretanto, alguns processos de WorkFlow não podem ser modelados utilizando a definição das *"Normalized WorkFlow nets"* devido às suas limitações de construção.

3.3 Lógica Linear

3.3.1 Definição

A lógica linear foi proposta em 1987 por Girard em [Girard 1987]. Com a existência de novos conectivos que formam um novo sistema lógico com várias características interessantes, como por exemplo a possibilidade de se interpretar um sequente como um estado de um sistema ou de uma fórmula como um recurso.

Na lógica linear, as proposições são consideradas como recursos que são consumidos ou produzidos a cada mudança de estado [Pradin-Chezalviel et al. 1999], ao passo que, na lógica clássica as proposições podem tomar valores *verdadeiro* ou *falso*. As principais diferenças entre a lógica clássica e a lógica linear são a inexistência das regras de enfraquecimento e de contração.

De acordo com a *Regra de Enfraquecimento*, se A implica B, então A e C implicam B [Gochet e Gribomont 1990]. Considerando um sistema em que A, B e C são recursos, a regra de enfraquecimento mostraria que recursos, neste caso C, poderiam aparecer e desaparecer sem nenhuma influência. Assim, se a partir de um recurso A pode-se produzir um recurso B (consumindo o recurso A), então o aparecimento de um recurso C altera o sistema, o que não é mostrado pela regra de enfraquecimento da lógica clássica [Soares 2004].

De acordo com a *Regra de Contração*, se A e A implicam B, então A implica B [Gochet e Gribomont 1990]. No contexto da lógica linear, utilizar essa regra seria o mesmo que aceitar que, havendo a necessidade de dois recursos A para produzir um recurso B, um desses recursos poderia ser dispensado, o que não é verdadeiro neste contexto, pois, se

houver apenas um recurso A, B não poderá mais ser produzido.

Outra importante diferença entre a lógica clássica e a lógica linear é que, enquanto na lógica clássica uma verdade é dita estável, na lógica linear o mesmo não acontece. Por exemplo, supondo que A e B sejam duas proposições da lógica clássica e que A seja verdadeira. Então, se A implica B, então B é deduzido. Assim, a proposição A continua sendo verdadeira mesmo após ter sido usada na dedução de B. Na lógica linear isso não é possível, pois se A foi consumido para produzir B, então A não é mais válido [Soares 2004].

3.3.2 Conectivos da Lógica Linear

A lógica linear introduz novos conectivos, como os conectivos *"par"*($\parr$), *"times"*($\otimes$), *"with"*(&), *"plus"*($\oplus$), *implicação linear* ($\multimap$), *"of course"* (!) e *"why not"* (?) [Girard 1987, Pradin-Chezalviel et al. 1999].

Estes conectivos estão divididos em três grupos:

- Os conectivos multiplicativos: *implicação linear* ($\multimap$), *"times"*($\otimes$) e *"par"*($\parr$).
- Os conectivos aditivos: *"with"*(&) e *"plus"*($\oplus$).
- Os conectivos exponenciais: *"of course"* (!) e *"why not"* (?).

No contexto deste trabalho, apenas dois conectivos da lógica linear serão explorados:

- O conectivo *times*, denotado pelo símbolo $\otimes$, representa a disponibilidade simultânea de recursos. Por exemplo, $A \otimes B$ representa a disponibilidade simultânea dos recursos A e B.
- O conectivo *implicação linear*, denotado pelo símbolo $\multimap$, representa uma mudança de estado. Por exemplo, $A \multimap B$ denota que consumindo A, B é produzido. Deve-se notar que após a produção de B, A não estará mais disponível.

3.3.3 Tradução das Redes de Petri em Fórmulas da Lógica Linear

Apesar de as redes de Petri poderem ser reescritas utilizando a lógica linear, as duas teorias não são equivalentes, e cada uma tem suas vantagens específicas [Champagnat et al. 2000]. As redes de Petri permitem o cálculo de invariantes de lugar e de transição, enquanto a lógica linear trata claramente cenários que envolvem a produção e consumo de recursos [Soares 2004].

A tradução de uma rede de Petri em fórmulas da lógica linear é apresentada em [Pradin-Chezalviel et al. 1999].

Uma marcação M é um monômio em $\otimes$, ou seja, uma marcação é representada por $M = A_1 \otimes A_2 \otimes ... \otimes A_k$ onde A_i são nomes de lugares. Por exemplo, considerando a rede

de Petri da Figura 3.4(a), tem-se que sua marcação inicial $M = P1$. Já a marcação inicial da rede de Petri da Figura 3.4(b) é dada por $M = P1 \otimes P2$.

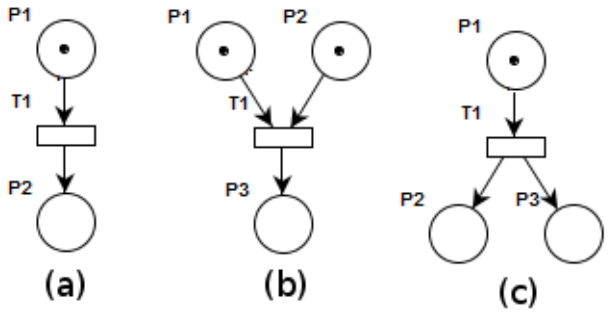

Figura 3.4: Redes de Petri para a exemplificação da tradução de redes de Petri em fórmulas da lógica linear.

Uma transição é uma expressão da forma $M_1 \multimap M_2$ onde M_1 e M_2 são marcações. Por exemplo, considerando a transição $t1$ da rede de Petri mostrada na Figura 3.4(a), tem-se que $t1 = P1 \multimap P2$. Para a transição $t1$ da rede de Petri da Figura 3.4(b), tem-se que $t1 = P1 \otimes P2 \multimap P3$. Já para a transição t1 da rede de Petri da Figura 3.4(c), tem-se que $t1 = P1 \multimap P2 \otimes P3$.

Um sequente $M, t_i \vdash M'$ representa um cenário em que M e M' são, respectivamente, a marcação inicial e final e t_i é uma lista de transições não ordenadas [Julia e Soares 2003]. Por exemplo, considerando a rede de Petri da Figura 3.4(a) e sua marcação final $P2$, tem-se o seguinte sequente linear $P1, P1 \multimap P2 \vdash P2$. Para a rede de Petri da Figura 3.4(b), considerando que sua marcação final será $P3$, tem-se o sequente $P1 \otimes P2, P1 \otimes P2 \multimap P3 \vdash P3$. Considerando a rede de Petri da Figura 3.4(c) e sua marcação final $P2 \otimes P3$, tem-se o sequente $P1, P1 \multimap P2 \otimes P3 \vdash P2 \otimes P3$.

3.3.4 Cálculo dos Sequentes

O sistema de dedução linear é similar ao sistema de dedução de Gentzen, proposto em 1934 [Gochet e Gribomont 1990]. Um sequente linear tem a forma $\Gamma \vdash \Delta$, onde Γ e Δ são conjuntos finitos de fórmulas, isto é, $\Gamma = \Gamma_1, \Gamma_2, ...\Gamma_n$ e $\Delta = \Delta_1, \Delta_2, ...\Delta_n$. O símbolo Γ é o antecedente da fórmula e o símbolo Δ o consequente.

Um sequente pode ser provado por meio de aplicações sucessivas de regras do cálculo de sequentes. De acordo com [Girault 1997], há equivalência entre a prova de sequentes da lógica linear e o problema de alcançabilidade em uma rede de Petri.

Uma árvore de prova da lógica linear é construída para provar se um dado sequente linear é ou não sintaticamente válido. A árvore de prova é lida de baixo para cima (bottom-up) e termina quando todas as folhas forem sequentes identidade (sequentes do tipo $A \vdash A$) [Julia e Soares 2003], considerando o caso em que o sequente é válido.

O presente trabalho considera apenas algumas regras da lógica linear. Assim, somente essas regras serão explicadas e terão suas aplicações exemplificadas. As demais regras do cálculo de sequentes da lógica linear podem ser encontradas em [Girault 1997] e em [Jean-Yves Girard e Regnier 1995]. As regras aqui apresentadas são utilizadas na construção das árvores de prova canônica no contexto das redes de Petri e, consequentemente, no contexto das *WorkFlow nets*. Para tanto, considere-se que F , G e H são fórmulas e que Γ e Δ são blocos de fórmulas da lógica linear. A seguir seguem as regras [Riviere et al. 2001]:

- A regra $\multimap_L$, $\dfrac{\Gamma \vdash F \quad \Delta, G \vdash H}{\Gamma, \Delta, F \multimap G \vdash H} \multimap_L$, expressa o disparo de uma transição e gera dois sequentes, tal que o sequente à direita representa o subsequente restante a ser provado e o sequente à esquerda representa as fichas consumidas pelo disparo da transição.
- A regra $\otimes_L$, $\dfrac{\Gamma, F, G \vdash H}{\Gamma, F \otimes G \vdash H} \otimes_L$ transforma uma marcação em uma lista de átomos.
- A regra $\otimes_R$, $\dfrac{\Gamma \vdash F \quad \Delta \vdash G}{\Delta, \Gamma \vdash F \otimes G} \otimes_R$ transforma um sequente do tipo $A, B \vdash A \otimes B$ em dois sequentes identidade $A \vdash A$ e $B \vdash B$.

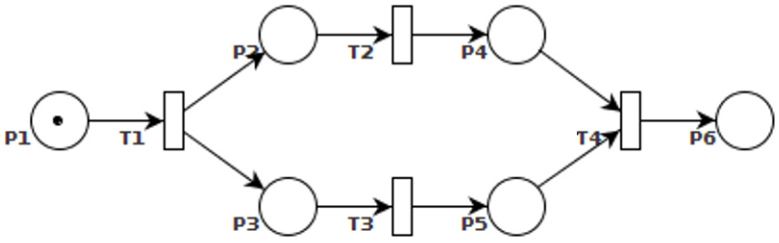

Figura 3.5: Rede de Petri para exemplificação da construção de uma árvore de prova canônica da lógica linear

Para a exemplificação da construção de uma árvore de prova canônica da lógica linear, considere-se a rede de Petri da Figura 3.5. Transformando as transições desta rede de Petri em fórmulas da lógica linear, tem-se que:

$$t_1 = P_1 \multimap P_2 \otimes P_3$$
$$t_2 = P_2 \multimap P_4$$
$$t_3 = P_3 \multimap P_5$$
$$t_4 = P_4 \otimes P_5 \multimap P_6$$

A marcação inicial desta rede de Petri é apenas $P1$. Assim, o sequente linear a ser provado é dado por $P1$, t_1 , t_2 , t_3 , $t_4 \vdash P6$, onde $P1$ e $P6$ são, respectivamente, a marcação inicial e final da rede de Petri da Figura 3.5. Aplicando as regras do cálculo

de sequentes a este sequente linear, é possível provar se o mesmo é ou não um sequente sintaticamente válido. A árvore de prova é a seguinte:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{P_4 \vdash P_4 \quad P_5 \vdash P_5}{P_4, P_5 \vdash P_4 \otimes P_5}\otimes_R \quad P_6 \vdash P_6}{P_3 \vdash P_3 \quad P_4, P_5, P_4 \otimes P_5 \multimap P_6 \vdash P_6}\multimap_L}{P_2 \vdash P_2 \quad P_3, P_4, P_3 \multimap P_5, t_4 \vdash P_6}\multimap_L}{P_2, P_3, P_2 \multimap P_4, t_3, t_4 \vdash P_6}\otimes_R}{P_1 \vdash P_1 \quad P_2 \otimes P_3, P_2 \multimap P_4, t_3, t_4 \vdash P_6}\multimap_L$$

$$P_1, P_1 \multimap P_2 \otimes P_3, t_2, t_3, t_4 \vdash P_6$$

Como todos os nós folha da árvore de prova acima são sequentes identidade, ou seja, sequentes do tipo $A \vdash A$, tem-se que o sequente linear $P1$, t_1 , t_2 , t_3 , $t_4 \vdash P6$ é sintaticamente válido.

3.3.5 Cálculo dos Sequentes com Tempo e Paralelismo

No contexto das redes de Petri t-temporais, em uma árvore de prova da lógica linear, cada disparo de transição pode gerar uma data simbólica associada a cada átomo (token), como mostrado em [Riviere et al. 2001].

O cálculo de datas em árvores de prova canônica é dado pelos seguintes passos:

- Determinar uma data de produção D_i para todas as fichas da marcação inicial;
- para cada instância da regra $\multimap_L$, calcular a data de disparo desta transição: isto é igual ao maior valor de data de produção dos átomos consumidos por esta transição, acrescido pela duração de sensibilização d_i associada à transição considerada;
- atualizar as datas de todos os átomos que foram consumidos e produzidos.

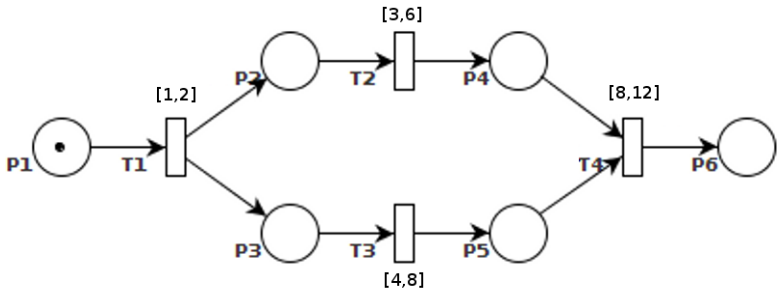

Figura 3.6: Rede de Petri t-temporal para exemplificação da construção de uma árvore de prova canônica da lógica linear com cálculo de datas.

Por exemplo, considerando a rede de Petri t-temporal da Figura 3.6 e $Seq = D_1 + d_1 + max(d_2, d_3) + d_4$, tem-se a seguinte árvore de prova com cálculo de datas:

$$\dfrac{\dfrac{\dfrac{\dfrac{P_4(D_1{+}d_1{+}d_2,Seq)\vdash P_4 \quad P_5(D_1{+}d_1{+}d_3,Seq)\vdash P_5}{P_4(D_1{+}d_1{+}d_2,Seq),P_5(D_1{+}d_1{+}d_3,Seq)\vdash P_4\otimes P_5}\otimes_R \quad P_6(Seq,.)\vdash P_6}{P_3(D_1{+}d_1,D_1{+}d_1{+}d_3)\vdash P_3 \quad P_4(D_1{+}d_1{+}d_2,.),P_5(D_1{+}d_1{+}d_3,.),P_4\otimes P_5\multimap P_6\vdash P_6}\multimap_L}{P_2(D_1{+}d_1,D_1{+}d_1{+}d_2)\vdash P_2 \quad P_3(D_1{+}d_1,.),P_4(D_1{+}d_1{+}d_2,.),P_3\multimap P_5,t_4\vdash P_6}\multimap_L}{P_2(D_1{+}d_1,.),P_3(D_1{+}d_1,.),P_2\multimap P_4,t_3,t_4\vdash P_6}\otimes_R}{P_1(D_1,D_1{+}d_1)\vdash P_1 \quad P_2(D_1{+}d_1,.)\otimes P_3(D_1{+}d_1,.),P_2\multimap P_4,t_3,t_4\vdash P_6}\multimap_L$$

$$P_1(D_1,.),P_1\multimap P_2\otimes P_3,t_2,t_3,t_4\vdash P_6$$

Em um modelo de rede de Petri t-temporal, toda duração de sensibilização d_i de uma transição t_i tem um valor que pertence a um intervalo de tempo $\Delta_i = [\delta_{imin}, \delta_{imax}]$. Logo, uma vez que as datas simbólicas computadas dependem de d_i, seus domínios também serão em função de intervalos de tempo. A Tabela 3.2 mostra os intervalos de datas simbólicas de produção e consumo de cada átomo da rede de Petri t-temporal da Figura 3.6.

Átomo	**Data de Produção**	**Data de Consumo**
P_1	D_1	$D_1 + d_1$
P_2	$D_1 + d_1$	$D_1 + d_1 + d_2$
P_3	$D_1 + d_1$	$D_1 + d_1 + d_3$
P_4	$D_1 + d_1 + d_2$	$D_1 + d_1 + max(d_2, d_3) + d_4$
P_5	$D_1 + d_1 + d_3$	$D_1 + d_1 + max(d_2, d_3) + d_4$
P_6	$D_1 + d_1 + max(d_2, d_3) + d_4$	*desconhecido*

Tabela 3.1: Datas simbólicas de produção e consumo dos átomos da rede de Petri t-temporal da Figura 3.6

Átomo	**Data de Produção**	**Data de Consumo**
P_1	$[D_{1_{min}}, D_{1_{max}}]$	$[D_{1_{min}} + d_{1_{min}}, D_{1_{max}} + d_{1_{max}}]$
P_2	$[D_{1_{min}} + d_{1_{min}}, D_{1_{max}} + d_{1_{max}}]$	$[D_{1_{min}} + d_{1_{min}} + d_{2_{min}}, D_{1_{max}} + d_{1_{max}} + d_{2_{max}}]$
P_3	$[D_{1_{min}} + d_{1_{min}}, D_{1_{max}} + d_{1_{max}}]$	$[D_{1_{min}} + d_{1_{min}} + d_{3_{min}}, D_{1_{max}} + d_{1_{max}} + d_{3_{max}}]$
P_4	$[D_{1_{min}} + d_{1_{min}} + d_{2_{min}}, D_{1_{max}} + d_{1_{max}} + d_{2_{max}}]$	$[D_{1_{min}} + d_{1_{min}} + max(d_{2_{min}}, d_{3_{min}}) + d_{4_{min}},$ $D_{1_{max}} + d_{1_{max}} + max(d_{2_{max}}, d_{3_{max}}) + d_{4_{max}}]$
P_5	$[D_{1_{min}} + d_{1_{min}} + d_{3_{min}}, D_{1_{max}} + d_{1_{max}} + d_{3_{max}}]$	$[D_{1_{min}} + d_{1_{min}} + max(d_{2_{min}}, d_{3_{min}}) + d_{4_{min}},$ $D_{1_{max}} + d_{1_{max}} + max(d_{2_{max}}, d_{3_{max}}) + d_{4_{max}}]$
P_6	$[D_{1_{min}} + d_{1_{min}} + max(d_{2_{min}}, d_{3_{min}}) + d_{4_{min}},$ $D_{1_{max}} + d_{1_{max}} + max(d_{2_{max}}, d_{3_{max}}) + d_{4_{max}}]$	*desconhecido*

Tabela 3.2: Intervalos de datas simbólicas de produção e consumo dos átomos da rede de Petri t-temporal da Figura 3.6

O tempo de execução da *WorkFlow net* é dado por $D_1 + d_1 + max(d_2, d_3) + d_4$ que corresponde à data de produção da ficha na posição $end(P_6)$. substituindo os valores numéricos, obtidos o intervalo de tempo será:$[13, 22]$.

3.3.6 Pova da Propriedade Soundness a partir do Calculo do Sequente

Deve-se notar que cada sequente linear tem apenas um átomo *Start*, que representa a marcação inicial da *WorkFlow net*. Isso acontece uma vez que, nesta abordagem, para analisar o critério de corretude *soundness*, apenas uma ficha no lugar *Start* é necessária e suficiente. Vale destacar que apenas uma ficha no lugar *Start* é necessária e suficiente pois, se não houver uma ficha no lugar *Start*, não haverá no sequente linear um átomo *Start* e a prova sintática do sequente não será realizada e se houver mais de uma ficha neste lugar, as demais fichas não serão utilizadas e ficarão remanescentes na árvore de prova. Assim, para a construção dos sequentes da lógica linear, sempre será considerada apenas uma ficha no lugar *Start*. A árvore de prova da lógica linear utilizada no contexto desse trabalho pode terminar de três formas:

- Quando todas as folhas da árvore de prova forem sequentes identidade;
- Quando o átomo End for produzido (ou seja, quando o sequente $End \vdash End$ aparecer na árvore de prova);
- Ou quando não houver nenhuma regra do cálculo de sequentes que possa ser aplicada.

Assim, a árvore de prova utilizada nesta abordagem é decidível, uma vez que cada transição da *WorkFlow net* aparece no máximo uma vez no sequente linear. Consequentemente, se o número de transições de uma *WorkFlow net* é finito, a construção da árvore de prova é decidível [Passos 2009].

Quando as árvores de prova para todos os cenários da *WorkFlow net* analisada estão construídas, elas devem ser analisadas seguindo os seguintes passos:

1. Para cada árvore de prova construída verificar se:

   (a) Apenas um átomo *End* foi produzido (isto é representado, na árvore de prova, pelo sequente identidade $End \vdash End$). Este fato prova o primeiro requisito para a verificação da propriedade *soundness*, isto é, que apenas uma ficha aparece no lugar *End* para o caso tratado.

   (b) Quando a prova está finalizada, não há nenhum átomo disponível para consumo na árvore de prova. Este fato prova que todos os lugares da WorkFlow net estão vazios para este caso quando sua ficha chega no lugar *End*, ou seja, o segundo requisito para afirmar que uma *WorkFlow net* é sound.

2. Considerando todos os cenários $Sc_1, Sc_2, ..., Sc_n$ da *WorkFlow net* analisada, cada transição $t \in T$ precisa aparecer em um cenário, pelo menos. Isto prova que todas as transições são disparadas, ou seja, nenhuma transição é morta.

Se as condições 1 e 2 acima são satisfeitas, a *WorkFlow net* analisada é *sound*.

Capítulo 4

Modelos de Cenários de Video Games

4.1 Estrutura de um Vídeo Game Clássico

O componente central de qualquer jogo, do ponto de vista da programação, é o *loop* do jogo. O *loop* do jogo permite que o jogo execute sem problemas, independentemente da entrada de um usuário ou da falta dela. Um *loop* de jogo pode ser representado numa forma altamente simplificada pelo seguinte pseudocódigo:

while user doesn't exit **do**
    check for user input
    run AI
    move enemies
    resolve collisions
    draw graphics
    play sounds
end while

O *loop* do jogo pode ser refinado e modificado com o progresso do desenvolvimento do jogo, mas a maioria dos jogos são baseados nesta ideia básica [Loki Software e Hall 2001].

Loops de jogo diferem, dependendo da plataforma para a qual são desenvolvidos. Por exemplo, jogos escritos para o MS-DOS e a maioria dos consoles podem explorar os recursos de processamento disponíveis sem restrições, por serem mono usuário e mono tarefa. No entanto, jogo para sistemas operacionais capazes de lidar com multitarefa, como por exemplo o Microsoft Windows, devem operar dentro dos limites do escalonador de processos. Alguns jogos modernos executam múltiplas *threads*, para que, por exemplo, o cálculo de AI (*artificial intelligence*) do personagem pode ser desacoplado da geração de movimento dentro do jogo. Essa abordagem tem a pequena desvantagem de aumentar a sobrecarga do processador, mas assim os jogos podem geralmente rodar de forma mais suave e mais eficiente em processadores *hyper-threading* ou *multicore*, assim como em plataformas com múltiplos processadores. Consoles como o Xbox 360 e PlayStation 3

já tem mais de um núcleo por processador, e podem executar mais de uma *thread* por núcleo.

4.1.1 Noção de Quest

No sentido mais geral, uma *quest* é uma "busca de um resultado específico". *Quests* típicas envolvem matar um determinado número de criaturas ou recolher uma lista de itens específicos. Algumas missões podem levar apenas alguns minutos ou horas para serem concluídas, enquanto outras podem demorar vários dias ou semanas. Frequentemente, quanto maior a recompensa, mais longa é a busca para concluir a *quest*, é comum exigir que os personagens atinjam um certo conjunto de condições prévias antes de serem autorizados a iniciar.

Questing é uma ferramenta usada em jogos de RPG(*Role-playing game*) para evitar que os jogadores fiquem em uma posição em que apenas realizaram uma ação repetitiva, como matar criaturas. Os jogadores podem realizar essa ação repetitiva a fim de ganhar novas habilidades e progredir para novas áreas ou em busca de dinheiro para comprar novos itens, como armaduras e equipamentos. Esse processo, vulgarmente conhecido como *"Grinding"*[1] pode retardar a progressão de um personagem no jogo e, finalmente, limitar a diversão do jogador. Ter um grande número de missões a enfrentar é visto geralmente como uma forma de oferecer variedade e de reduzir a necessidade do *"grinding"*.

Uma *side-quest*[2] é uma seção opcional do game e é comumente encontrada em jogos de *RPG*. É uma missão menor, dentro de uma grande história, e pode ser usada como um meio para fornecer estrutura não-linear. Como regra geral, a conclusão de *side-quests* não são essenciais para a conclusão do jogo, mas podem trazer vários benefícios para os personagens.

Quests são geralmente agrupadas em diversas categorias como:

- **Kill quests** Uma *kill quest* direciona o personagem a matar um número específico de criaturas ou um personagem específico. Estes tipos de *quests* muitas vezes exigem que o personagem retorne com uma prova de seu trabalho, tais como os dentes de animais, ou a cabeça de um personagem específico. Existem também *quests* para matar os outros jogadores.
- **Combo quests** As *combo quests* exigem que o jogador ataque determinados inimigos ou estruturas com uma combinação de ataques até que o alcance um número necessário de combos. Inimigos nessas *quests* geralmente são imortais ou infinito em número, e continuaram aparecendo até que o jogador seja bem sucedido, alcançando o número de combos necessários.

[1] é um termo usado em jogos de vídeo para descrever o processo de se engajar em tarefas repetitivas e/ou entediantes não pertencentes à linha de história do jogo

[2] missão paralela

- **Delivery quests** As *delivery quests* envolvem enviar o personagem para entregar um item de um local para outro, geralmente o personagem necessita encontrar o item anteriormente. Estas missões são desafiadoras, pedindo que o personagem viagem por um terreno desconhecido ou perigoso, às vezes enquanto enfrenta um limite de tempo.
- **Gather quests** As *gather quests* , também conhecido como *collection quests*, requerem que o personagem colete uma série de itens. Estas podem ser recolhidas a partir de um local ou ambiente, ou exigir que o personagem mate criaturas a fim de coletar os itens necessários. A busca pode também exigir que o personagem recolha uma série de itens diferentes, por exemplo, para montar um dispositivo.
- **Escort quests** As *escort quests* são caracterizadas por matar monstros para proteger um aliado. A missão de escolta típica envolveria proteger um aliado enquanto esse atravessa uma área infestada de monstros. A maior parte do tempo a *quest* exigirá que o jogador destrua múltiplos inimigos para garantir a segurança do aliado.
- **Híbridos** Elementos dos tipos acima podem ser combinados para criar *quests* mais complexas. Por exemplo, uma *quest* pode exigir que o jogador encontre as peças necessárias para montar uma arma (*gather quest*) e usá-la para matar um inimigo específico (*kill quest*). *Quests* híbridas também podem incluir quebra-cabeças e charadas.

A noção de atividades numa *quest* consiste da composição de um conjunto de ações elementares. Ações elementares representam então ações básicas que um jogador pode executar como andar, correr, pular, entre outras. Uma *quest*, dependendo do tamanho, pode ser sinônimo de um nível do jogo mas um nível também pode ser composto de um conjunto de *quests*.

O video game *The Legend of Zelda* será usado para ilustrar a abordagem apresentada neste trabalho. É uma série de vídeo games com uma história de fantasia misturada com muita ação e aventura, criada pelos japoneses Shigeru Miyamoto e Takashi Tezuka. Considerado uma das franquias mais importantes da Nintendo, a sua jogabilidade consiste de uma mistura de ação, aventura e quebra-cabeças. A série é centrada em Link, o personagem principal controlado pelo jogador e protagonista. Link muitas vezes tem a tarefa de resgatar a Princesa Zelda no cenário mais comum da série.

Em *The Legend of Zelda: Oracle of Ages*, a Árvore Maku é destruída no passado sob ordens do vilão Veran; Link, usa então um portal do tempo para viajar ao passado e evitar que tal evento aconteça. A árvore Maku indica a Link que ele terá que coletar as oito Essências do Tempo para derrotar Veran. O jogador passa a maior parte de seu tempo tentando encontrar as oito Essências do Tempo, cada uma escondida em um calabouço que contém inimigos e quebra-cabeças.

4.2 Definição de Cenários de Video Games

4.2.1 Modelo de quest

Numa *quest*, uma atividade simples é a execução básica de um conjunto de operações elementares que podem ser associadas a uma transição de uma *WorkFlow net*. Por exemplo, a Figura 4.1 mostra um exemplo de atividade simples, representada pelo disparo da transição t_0 a qual ocorre quando o monstro for morto. Na verdade, matar o monstro será constituído por um conjunto de ações elementares que o jogador executará para realizar a atividade correspondente.

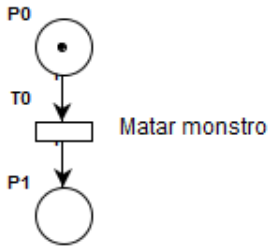

Figura 4.1: Atividade Simples.

Atividades sequenciais são atividades executadas em ordem sucessiva, havendo, claramente, dependência entre elas. Nos modelos de *quest*, atividades sequenciais serão utilizadas para comunicar uma ideia de continuidade na história, criando assim uma "linha guia"[3] para conclusão da *quest*.

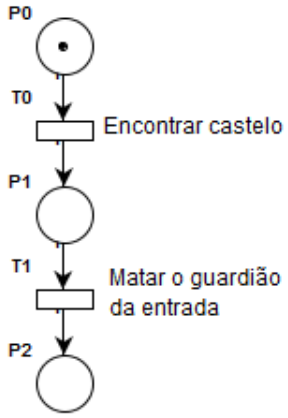

Figura 4.2: Atividade Sequencial.

A Figura 4.2 mostra um exemplo de atividades sequenciais em que o jogador necessita encontrar o castelo e, em seguida, matar o guardião que está localizado na entrada. É claro que o disparo da transição t_1 (matar o guardião) só ocorrerá depois de finalizar o disparo da transição t_0 (encontrar o castelo).

[3]do inglês, *guideline*

Numa *quest* atividades paralelas serão representadas por um tipo de roteiro em que as atividades serão executadas simultaneamente ou em ordem arbitrária. Nesse caso, as atividades serão executadas sem que o resultado de uma interfira no resultado da outra. Sendo assim, o jogador poderá escolher qual atividade será realizada primeiro.

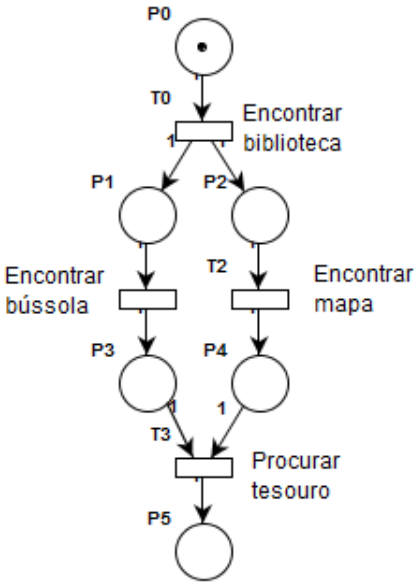

Figura 4.3: Atividade Paralela.

A Figura 4.3 mostra um exemplo de atividades paralelas. Após o jogador encontrar a biblioteca, inicia a busca pelo mapa e pela bússola; como não sabe onde esses artefatos estão escondidos, irá procurar em cada baú, gaveta, estante, até eventualmente encontrá-los, caracterizando assim uma busca em paralelo. Ao encontrar um dos itens, irá disparar a transição correspondente (encontrar bússola ou encontrar mapa) Após encontrar ambos artefatos será possível continuar a *quest* iniciando a busca pelo tesouro.

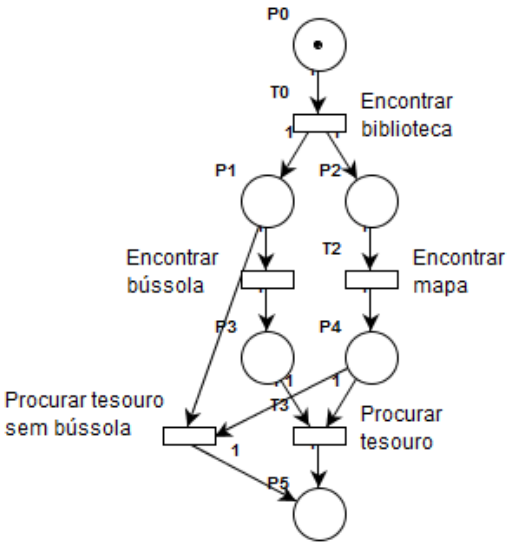

Figura 4.4: Atividade Opcional

Uma atividade opcional será associada a um tipo de roteiro seletivo que permite completar ou não uma *quest*. A Figura 4.4 mostra um exemplo de atividade opcional representada pela transição t_1. É claro que a atividade "procurar o tesouro sem bússola", representada pela transição t_1, não é obrigatória para completar a *quest* (produzir uma ficha em P_5).

Formalmente, uma *quest* pode ser vista como um grupo de atividades que devem ser executadas em sequência ou simultaneamente. Um modelo de *quest* consistira então da combinação de diversos roteiros de atividade a fim de expressar uma determinada missão.

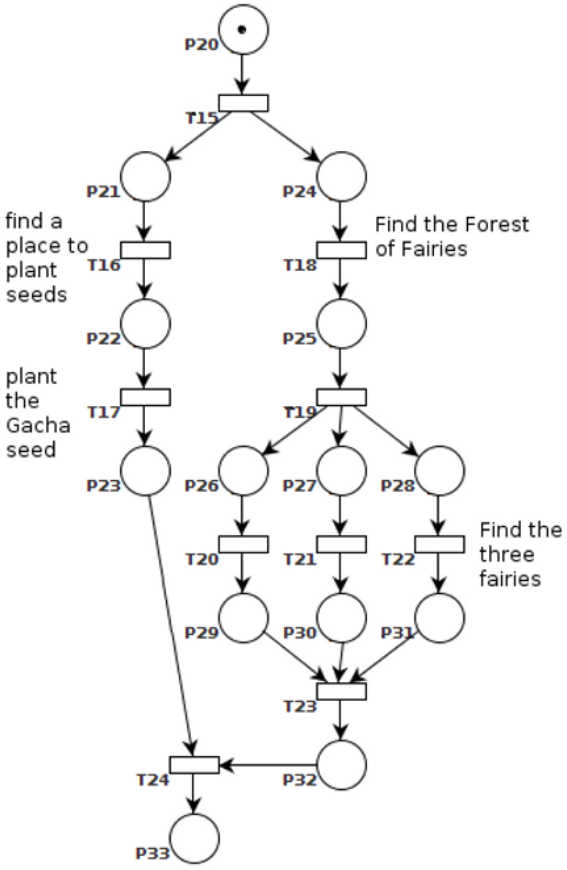

Figura 4.5: Quest 7 sound.

A *WorkFlow net* da Figura 4.5 representa um exemplo de *quest* que Link deve completar no Zelda. As atividades são associadas com as transições e as condições com os lugares. O lugar inicial P_{20} representa o início da *quest* e o último lugar P_{33} o objetivo a ser alcançado para finalizar a *quest*.

A ficha na posição inicial representa Link e o início da *quest*. "Find a place to plant seeds" e "Plant the Gacha seed" são atividades que devem ser executadas em sequência. As transições T_{20}, T_{21} e T_{22} representam cada fada a ser encontrada. Essas atividades seguem o roteiro paralelo e podem ser executadas simultaneamente. Isso significa que Link pode procurar as três fadas ao mesmo tempo. Antes de procurar pelas fadas, Link deve encontrar a floresta. Essa atividade está associada com a transição T_{18}. As atividade relacionadas a semente e à busca pelas fadas podem ser executadas em paralelo. Tal fato

é representado pela transição T_{15} que inicia duas evoluções em paralelo. Isso significa que Link pode procurar pelas fadas e pela semente ao mesmo tempo. No final da *quest*, após ter completado todas atividades, Link concluirá a *quest* chegando ao lugar de término.

4.2.2 Modelo de uma rede de quests ou de níveis

Formalmente, um jogo pode ser visto como um grupo de *quests* e/ou de níveis que devem ser executados em sequência ou simultaneamente. Uma rede de *quests* e/ou níveis consistirá então da combinação de diversos roteiros de *quests* e/ou níveis que caracterizará os grandes objetivos do jogo a serem alcançados.

A Figura 4.6 mostra um exemplo de jogo com um conjunto de cinco *quests*. Após o jogador concluir a *quest* 1, as *quests* 2, 3 e 4 serão liberadas. O próprio jogador poderá escolher a ordem de realização das *quests*. A *quest* 5 apenas será liberada após a conclusão da *quest* 2. A *quest* 4 é considerada como uma *quest* opcional pois o jogador pode terminar o jogo sem passar por ela ao disparar a transição T_7.

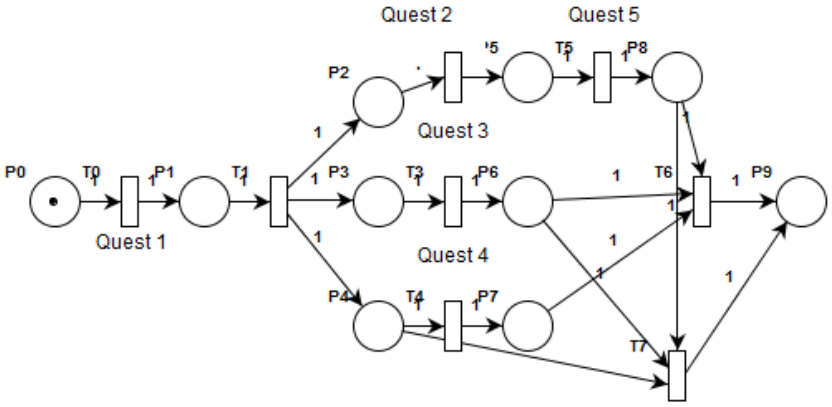

Figura 4.6: Rede de *quests*.

4.2.3 Modelo de Quest com Recursos

Durante uma *quest* no jogo, alguns itens pode ser encontrados. Do ponto de vista de um modelo de *quest* tais itens serão vistos como recursos discretos. No contexto das redes de Petri, um recurso representa geralmente um meio técnico ou humano necessário à realização de uma atividade. Diversos tipos de recursos podem então ser considerados. Um recurso será considerado como renovável se depois de ter sido alocado à uma ou diversas atividades, ele é disponível novamente na mesma quantidade. No caso contrário, o recurso é consumível (matéria bruta, dinheiro) e sua utilização no tempo é limitada.

No caso de video games, recursos são itens que uma vez encontrados irão permanecer ou não com o jogador. No Zelda por exemplo existem vários itens como por exemplo espada, escudo, pá, bracelete, os quais o jogador precisa para completar as *quests*.

Alguns desses itens, uma vez encontrados permanecerão com o jogador até o fim do jogo, como a espada por exemplo, e serão considerados como recursos discretos renováveis. Outros, serão utilizados para realizar atividades específicas (uma chave por exemplo utilizada para abrir uma porta) e serão considerados como recursos discretos consumíveis.

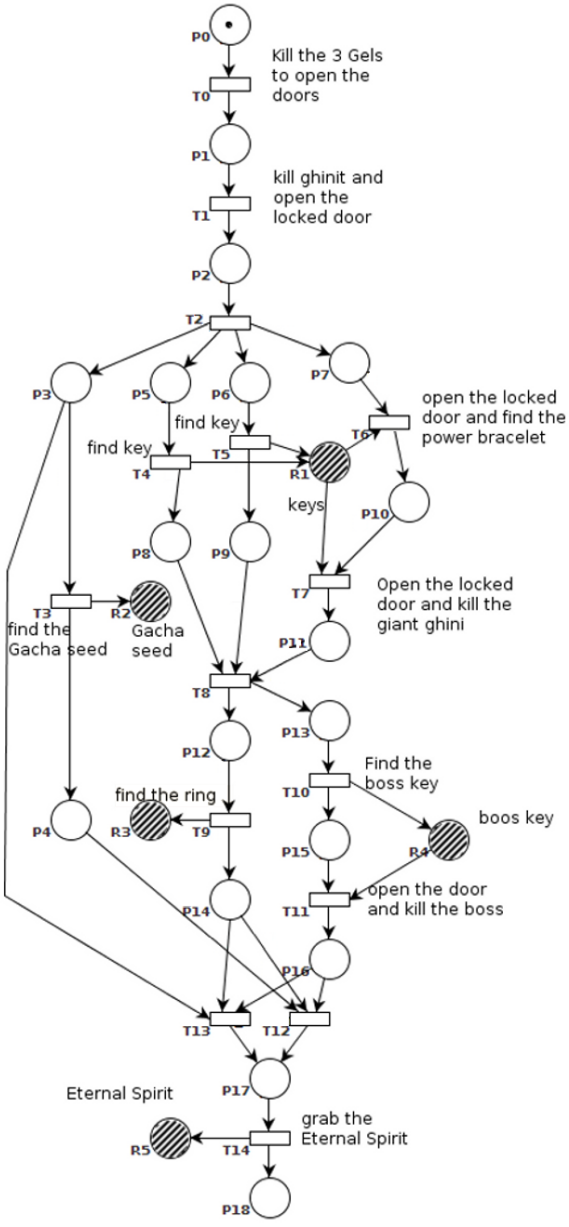

Figura 4.7: Quest 6

Numa *WorkFlow net*, um recurso discreto será representado por uma ficha num lugar específico que representará a disponibilidade do recurso. A *WorkFlow net* na Figura 4.7 representa a *quest* número 6 que deve ser realizada anteriormente à *quest* número 7 representada na Figura 4.5. Uma das principais atividades desta *quest* é encontrar o item "Gacha seed". Os lugares de recursos na *WorkFlow net* da Figura 4.7 são representados pelos lugares R_1, R_2, R_3, R_4 e R_5.

Por exemplo, para abrir a porta (transição T_6) uma chave é necessária e pode ser pro-

duzida pela atividade "Find a key" (transição T_4 ou T_5). Todos os recursos representados na *quest* número 6 são recursos consumíveis, para não sobrecarregar os modelos de *quest* os recursos renováveis não foram representados.

4.2.4 Modelo de Rede de Quest ou Níveis com Recursos

Alguns dos itens encontrados numa *quest* específica podem ser necessários ao bom andamento de *quests* futuras. Assim, alguns dos recursos discretos produzidos em modelos de *quest* específicos, deverão ser repassados para outras *quests* do jogo. Num modelo de rede de *quest* ou de nível, a comunicação de recursos entre *quests* dependerá de um lugar de comunicação específico. O arco de entrada do lugar de comunicação será também um arco de saída da transição associada à *quest* onde o recurso será produzido. O arco de saída do lugar de comunicação será também um arco de entrada da transição associada à *quest* que utilizará o recurso produzido. A Figura 4.8 apresenta um exemplo onde o recurso vindo da *quest* número 1 representado pela transição T_1 alimenta o lugar R_1, o qual é responsável por representar o recurso que será consumido pela *quest* 2 representada pela transição T_2.

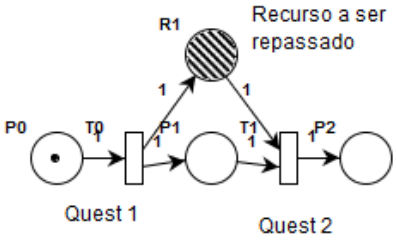

Figura 4.8: Transmissão de recurso entre quests

Para manter a propriedade de *soudness* nas *WorkFlow nets*, deve se adicionar, dentro de um mesmo modelo de *quest*, arcos artificiais que ligam os lugares que representam os recursos disponíveis (que serão repassados as outras *quests*) à última transição da *WorkFlow net* correspondente. Assim, ao finalizar uma *quest*, para verificar a propriedade de *soundness*, todos os lugares da *WorkFlow net*, exceto o lugar de fim, deverão estar vazios. A Figura 4.9 exibe um exemplo de repasse de recursos.

Finalmente, dentro do modelo de *quest* onde será repassado um recurso, um lugar que representa o recurso disponível deverá ser criado tendo como arco de entrada um arco saindo da primeira transição da *WorkFlow net* correspondente. Assim, ao iniciar uma nova *quest*, todos os recursos adquiridos em *quests* anteriores serão enviados à *quest* em curso. A Figura 4.10 demostra como a transição T_1 foi conectada ao recurso R_1, assim disponibilizando esse recurso para consumo.

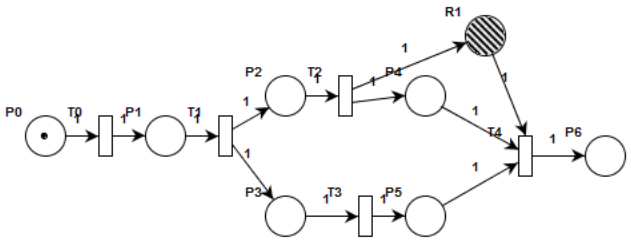

Figura 4.9: Envio do Recurso ao finalizar a quest

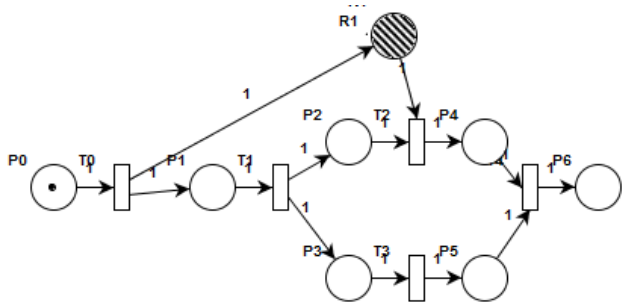

Figura 4.10: Recebendo o recurso ao iniciar a quest

4.2.5 Modelo de Quest temporal

O principal objetivo do planejamento temporal apresentado neste trabalho é calcular janelas de datas (intervalo de tempo simbólico) para cada *quest*. Assim, será possível prever a o tempo de jogo.

A fim de representar a duração de cada *quest*, um intervalo de tempo pode ser atribuído a cada atividade, representando um tempo mínimo e um tempo máximo para a duração das atividades.

O modelo de uma *quest* será então representado por uma rede de Petri t-temporal (t-Time). Pelo fato do modelo de *quest* ser aplicado na fase de *game design*, não será ainda possível definir as durações das atividades. Os intervalos de tempo associados a cada atividade serão então dados por meio de durações simbólicas, ao invés de dados numéricos. Por meio das árvores de prova da lógica linear, datas simbólicas de execução da *quest* poderão então ser derivadas.

Finalmente, os modelos temporizados serão *WorkFlow nets* sem ciclos (sem roteiros interativos). Os cenários repetidos não serão considerados numa forma explícita. No caso de um jogador não conseguir completar uma *quest*, o mesmo terá de repeti-la. O tempo de jogabilidade será calculado por meio da somatória do tempo de todas *quests* efetuadas (inclusive as *quests* repetidas).

A Figura 5.6 representa um exemplo de um modelo de *quest* t-temporal onde os intervalos de tempo associados às transições são dados por duração simbólica.

Capítulo 5

Análise de Cenário de Jogos

5.1 Análise Qualitativa de uma Quest

A principal propriedade que será verificada é a propriedade de *soudness* que garante que a *quest* poderá ser finalizada e que todas as atividades (obrigatórias e optativas) poderão ser realizadas.

5.1.1 Simulação

Verificar a propriedade de *soudness*, que garante a integridade do cenário da *quest*, usando técnicas de simulação é problemático sedo necessário considerar todas as sequências de disparos possíveis, pelo fato da ordem dos disparos ser um fator gera um crescimento exponencial de sequências.

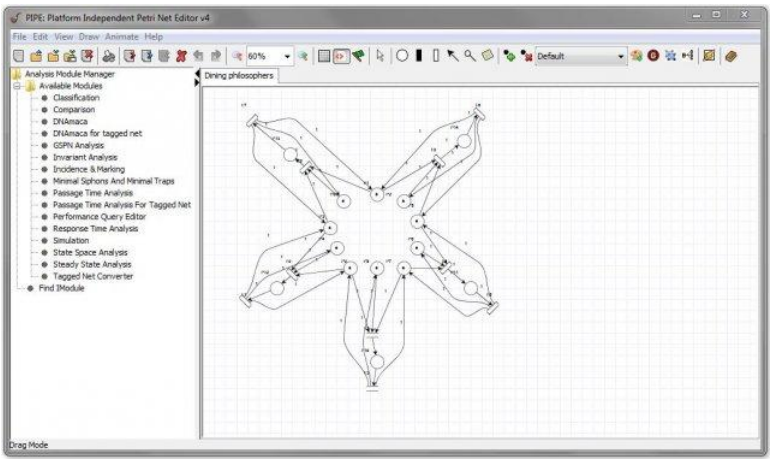

Figura 5.1: Platform Independent Petri Net Editor

Alguns *softwares* como o PIPE (*Platform Independent Petri Net Editor*) facilitam muito a criação dos modelos baseados em redes de Petri. A Figura 5.1 mostra a tela de edição das redes Petri em particular, PIPE possui um conjunto de módulos extensíveis que permite realizar diversos tipos de análise no modelo editado. É um projeto *open*

source hospedado no *sourceforge.net*, o que possibilita a criação de novos módulos como também possibilidades de modificações no código do *software*.

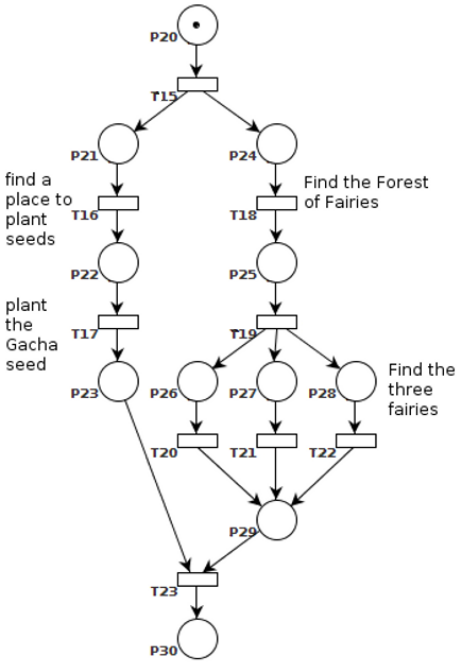

Figura 5.2: Quest 7 not sound

A Figura 5.2 apresenta um exemplo de *quest* não *sound*, por meio de simulação, pode se identificar tal falha.

As possíveis sequências de disparo para essa *WorkFlow net* produzidas pelo PIPE são:

1. T_{15},T_{16},T_{17},T_{18},T_{19},T_{20},T_{21},T_{22},T_{23} => P_{29},P_{29},P_{30}
2. T_{15},T_{16},T_{17},T_{18},T_{19},T_{20},T_{22},T_{21},T_{23} => P_{29},P_{29},P_{30}
3. T_{15},T_{16},T_{17},T_{18},T_{19},T_{21},T_{20},T_{22},T_{23} => P_{29},P_{29},P_{30}
4. T_{15},T_{16},T_{17},T_{18},T_{19},T_{22},T_{21},T_{20},T_{23} => P_{29},P_{29},P_{30}
5. T_{15},T_{16},T_{17},T_{18},T_{19},T_{21},T_{22},T_{20},T_{23} => P_{29},P_{29},P_{30}
6. T_{15},T_{16},T_{17},T_{18},T_{19},T_{22},T_{20},T_{21},T_{23} => P_{29},P_{29},P_{30}
7. T_{15},T_{16},T_{18},T_{17},T_{19},T_{20},T_{21},T_{22},T_{23} => P_{29},P_{29},P_{30}
8. T_{15},T_{16},T_{18},T_{17},T_{19},T_{20},T_{22},T_{21},T_{23} => P_{29},P_{29},P_{30}
9. T_{15},T_{16},T_{18},T_{17},T_{19},T_{21},T_{20},T_{22},T_{23} => P_{29},P_{29},P_{30}
10. T_{15},T_{16},T_{18},T_{17},T_{19},T_{22},T_{21},T_{20},T_{23} => P_{29},P_{29},P_{30}
11. T_{15},T_{16},T_{18},T_{17},T_{19},T_{21},T_{22},T_{20},T_{23} => P_{29},P_{29},P_{30}
12. T_{15},T_{16},T_{18},T_{17},T_{19},T_{22},T_{20},T_{21},T_{23} => P_{29},P_{29},P_{30}
13. T_{15},T_{16},T_{18},T_{19},T_{17},T_{20},T_{21},T_{22},T_{23} => P_{29},P_{29},P_{30}

14. T_{15},T_{16},T_{18},T_{19},T_{17},T_{20},T_{22},T_{21},T_{23} => P_{29},P_{29},P_{30}
15. T_{15},T_{16},T_{18},T_{19},T_{17},T_{21},T_{20},T_{22},T_{23} => P_{29},P_{29},P_{30}
16. T_{15},T_{16},T_{18},T_{19},T_{17},T_{22},T_{21},T_{20},T_{23} => P_{29},P_{29},P_{30}
17. T_{15},T_{16},T_{18},T_{19},T_{17},T_{21},T_{22},T_{20},T_{23} => P_{29},P_{29},P_{30}
18. T_{15},T_{16},T_{18},T_{19},T_{17},T_{22},T_{20},T_{21},T_{23} => P_{29},P_{29},P_{30}
19. T_{15},T_{16},T_{18},T_{19},T_{20},T_{17},T_{21},T_{22},T_{23} => P_{29},P_{29},P_{30}
20. T_{15},T_{16},T_{18},T_{19},T_{20},T_{17},T_{22},T_{21},T_{23} => P_{29},P_{29},P_{30}
21. T_{15},T_{16},T_{18},T_{19},T_{21},T_{17},T_{20},T_{22},T_{23} => P_{29},P_{29},P_{30}
22. T_{15},T_{16},T_{18},T_{19},T_{22},T_{17},T_{21},T_{20},T_{23} => P_{29},P_{29},P_{30}
23. T_{15},T_{16},T_{18},T_{19},T_{21},T_{17},T_{22},T_{20},T_{23} => P_{29},P_{29},P_{30}
24. T_{15},T_{16},T_{18},T_{19},T_{22},T_{17},T_{20},T_{21},T_{23} => P_{29},P_{29},P_{30}
25. T_{15},T_{16},T_{18},T_{19},T_{20},T_{21},T_{17},T_{22},T_{23} => P_{29},P_{29},P_{30}
26. T_{15},T_{16},T_{18},T_{19},T_{20},T_{22},T_{17},T_{21},T_{23} => P_{29},P_{29},P_{30}
27. T_{15},T_{16},T_{18},T_{19},T_{21},T_{20},T_{17},T_{22},T_{23} => P_{29},P_{29},P_{30}
28. T_{15},T_{16},T_{18},T_{19},T_{22},T_{21},T_{17},T_{20},T_{23} => P_{29},P_{29},P_{30}
29. T_{15},T_{16},T_{18},T_{19},T_{21},T_{22},T_{17},T_{20},T_{23} => P_{29},P_{29},P_{30}
30. T_{15},T_{16},T_{18},T_{19},T_{22},T_{20},T_{17},T_{21},T_{23} => P_{29},P_{29},P_{30}
31. T_{15},T_{16},T_{18},T_{19},T_{20},T_{21},T_{22},T_{17},T_{23} => P_{29},P_{29},P_{30}
32. T_{15},T_{16},T_{18},T_{19},T_{20},T_{22},T_{21},T_{17},T_{23} => P_{29},P_{29},P_{30}
33. T_{15},T_{16},T_{18},T_{19},T_{21},T_{20},T_{22},T_{17},T_{23} => P_{29},P_{29},P_{30}
34. T_{15},T_{16},T_{18},T_{19},T_{22},T_{21},T_{20},T_{17},T_{23} => P_{29},P_{29},P_{30}
35. T_{15},T_{16},T_{18},T_{19},T_{21},T_{22},T_{20},T_{17},T_{23} => P_{29},P_{29},P_{30}
36. T_{15},T_{16},T_{18},T_{19},T_{22},T_{20},T_{21},T_{17},T_{23} => P_{29},P_{29},P_{30}
37. T_{15},T_{18},T_{16},T_{17},T_{19},T_{20},T_{21},T_{22},T_{23} => P_{29},P_{29},P_{30}
38. T_{15},T_{18},T_{16},T_{17},T_{19},T_{20},T_{22},T_{21},T_{23} => P_{29},P_{29},P_{30}
39. T_{15},T_{18},T_{16},T_{17},T_{19},T_{21},T_{20},T_{22},T_{23} => P_{29},P_{29},P_{30}
40. T_{15},T_{18},T_{16},T_{17},T_{19},T_{22},T_{21},T_{20},T_{23} => P_{29},P_{29},P_{30}
41. T_{15},T_{18},T_{16},T_{17},T_{19},T_{21},T_{22},T_{20},T_{23} => P_{29},P_{29},P_{30}
42. T_{15},T_{18},T_{16},T_{17},T_{19},T_{22},T_{20},T_{21},T_{23} => P_{29},P_{29},P_{30}
43. T_{15},T_{18},T_{16},T_{19},T_{17},T_{20},T_{21},T_{22},T_{23} => P_{29},P_{29},P_{30}
44. T_{15},T_{18},T_{16},T_{19},T_{17},T_{20},T_{22},T_{21},T_{23} => P_{29},P_{29},P_{30}

45. T_{15},T_{18},T_{16},T_{19},T_{17},T_{21},T_{20},T_{22},T_{23} => P_{29},P_{29},P_{30}
46. T_{15},T_{18},T_{16},T_{19},T_{17},T_{22},T_{21},T_{20},T_{23} => P_{29},P_{29},P_{30}
47. T_{15},T_{18},T_{16},T_{19},T_{17},T_{21},T_{22},T_{20},T_{23} => P_{29},P_{29},P_{30}
48. T_{15},T_{18},T_{16},T_{19},T_{17},T_{22},T_{20},T_{21},T_{23} => P_{29},P_{29},P_{30}
49. T_{15},T_{18},T_{16},T_{19},T_{20},T_{17},T_{21},T_{22},T_{23} => P_{29},P_{29},P_{30}
50. T_{15},T_{18},T_{16},T_{19},T_{20},T_{17},T_{22},T_{21},T_{23} => P_{29},P_{29},P_{30}
51. T_{15},T_{18},T_{16},T_{19},T_{21},T_{17},T_{20},T_{22},T_{23} => P_{29},P_{29},P_{30}
52. T_{15},T_{18},T_{16},T_{19},T_{22},T_{17},T_{21},T_{20},T_{23} => P_{29},P_{29},P_{30}
53. T_{15},T_{18},T_{16},T_{19},T_{21},T_{17},T_{22},T_{20},T_{23} => P_{29},P_{29},P_{30}
54. T_{15},T_{18},T_{16},T_{19},T_{22},T_{17},T_{20},T_{21},T_{23} => P_{29},P_{29},P_{30}
55. T_{15},T_{18},T_{16},T_{19},T_{20},T_{21},T_{17},T_{22},T_{23} => P_{29},P_{29},P_{30}
56. T_{15},T_{18},T_{16},T_{19},T_{20},T_{22},T_{17},T_{21},T_{23} => P_{29},P_{29},P_{30}
57. T_{15},T_{18},T_{16},T_{19},T_{21},T_{20},T_{17},T_{22},T_{23} => P_{29},P_{29},P_{30}
58. T_{15},T_{18},T_{16},T_{19},T_{22},T_{21},T_{17},T_{20},T_{23} => P_{29},P_{29},P_{30}
59. T_{15},T_{18},T_{16},T_{19},T_{21},T_{22},T_{17},T_{20},T_{23} => P_{29},P_{29},P_{30}
60. T_{15},T_{18},T_{16},T_{19},T_{22},T_{20},T_{17},T_{21},T_{23} => P_{29},P_{29},P_{30}
61. T_{15},T_{18},T_{16},T_{19},T_{20},T_{21},T_{22},T_{17},T_{23} => P_{29},P_{29},P_{30}
62. T_{15},T_{18},T_{16},T_{19},T_{20},T_{22},T_{21},T_{17},T_{23} => P_{29},P_{29},P_{30}
63. T_{15},T_{18},T_{16},T_{19},T_{21},T_{20},T_{22},T_{17},T_{23} => P_{29},P_{29},P_{30}
64. T_{15},T_{18},T_{16},T_{19},T_{22},T_{21},T_{20},T_{17},T_{23} => P_{29},P_{29},P_{30}
65. T_{15},T_{18},T_{16},T_{19},T_{21},T_{22},T_{20},T_{17},T_{23} => P_{29},P_{29},P_{30}
66. T_{15},T_{18},T_{16},T_{19},T_{22},T_{20},T_{21},T_{17},T_{23} => P_{29},P_{29},P_{30}
67. T_{15},T_{18},T_{19},T_{16},T_{17},T_{20},T_{21},T_{22},T_{23} => P_{29},P_{29},P_{30}
68. T_{15},T_{18},T_{19},T_{16},T_{17},T_{20},T_{22},T_{21},T_{23} => P_{29},P_{29},P_{30}
69. T_{15},T_{18},T_{19},T_{16},T_{17},T_{21},T_{20},T_{22},T_{23} => P_{29},P_{29},P_{30}
70. T_{15},T_{18},T_{19},T_{16},T_{17},T_{22},T_{21},T_{20},T_{23} => P_{29},P_{29},P_{30}
71. T_{15},T_{18},T_{19},T_{16},T_{17},T_{21},T_{22},T_{20},T_{23} => P_{29},P_{29},P_{30}
72. T_{15},T_{18},T_{19},T_{16},T_{17},T_{22},T_{20},T_{21},T_{23} => P_{29},P_{29},P_{30}
73. T_{15},T_{18},T_{19},T_{16},T_{20},T_{17},T_{21},T_{22},T_{23} => P_{29},P_{29},P_{30}
74. T_{15},T_{18},T_{19},T_{16},T_{20},T_{17},T_{22},T_{21},T_{23} => P_{29},P_{29},P_{30}
75. T_{15},T_{18},T_{19},T_{16},T_{21},T_{17},T_{20},T_{22},T_{23} => P_{29},P_{29},P_{30}

76. T_{15},T_{18},T_{19},T_{16},T_{22},T_{17},T_{21},T_{20},T_{23} => P_{29},P_{29},P_{30}
77. T_{15},T_{18},T_{19},T_{16},T_{21},T_{17},T_{22},T_{20},T_{23} => P_{29},P_{29},P_{30}
78. T_{15},T_{18},T_{19},T_{16},T_{22},T_{17},T_{20},T_{21},T_{23} => P_{29},P_{29},P_{30}
79. T_{15},T_{18},T_{19},T_{16},T_{20},T_{21},T_{17},T_{22},T_{23} => P_{29},P_{29},P_{30}
80. T_{15},T_{18},T_{19},T_{16},T_{20},T_{22},T_{17},T_{21},T_{23} => P_{29},P_{29},P_{30}
81. T_{15},T_{18},T_{19},T_{16},T_{21},T_{20},T_{17},T_{22},T_{23} => P_{29},P_{29},P_{30}
82. T_{15},T_{18},T_{19},T_{16},T_{22},T_{21},T_{17},T_{20},T_{23} => P_{29},P_{29},P_{30}
83. T_{15},T_{18},T_{19},T_{16},T_{21},T_{22},T_{17},T_{20},T_{23} => P_{29},P_{29},P_{30}
84. T_{15},T_{18},T_{19},T_{16},T_{22},T_{20},T_{17},T_{21},T_{23} => P_{29},P_{29},P_{30}
85. T_{15},T_{18},T_{19},T_{16},T_{20},T_{21},T_{22},T_{17},T_{23} => P_{29},P_{29},P_{30}
86. T_{15},T_{18},T_{19},T_{16},T_{20},T_{22},T_{21},T_{17},T_{23} => P_{29},P_{29},P_{30}
87. T_{15},T_{18},T_{19},T_{16},T_{21},T_{20},T_{22},T_{17},T_{23} => P_{29},P_{29},P_{30}
88. T_{15},T_{18},T_{19},T_{16},T_{22},T_{21},T_{20},T_{17},T_{23} => P_{29},P_{29},P_{30}
89. T_{15},T_{18},T_{19},T_{16},T_{21},T_{22},T_{20},T_{17},T_{23} => P_{29},P_{29},P_{30}
90. T_{15},T_{18},T_{19},T_{16},T_{22},T_{20},T_{21},T_{17},T_{23} => P_{29},P_{29},P_{30}
91. T_{15},T_{18},T_{19},T_{20},T_{16},T_{17},T_{21},T_{22},T_{23} => P_{29},P_{29},P_{30}
92. T_{15},T_{18},T_{19},T_{20},T_{16},T_{17},T_{22},T_{21},T_{23} => P_{29},P_{29},P_{30}
93. T_{15},T_{18},T_{19},T_{21},T_{16},T_{17},T_{20},T_{22},T_{23} => P_{29},P_{29},P_{30}
94. T_{15},T_{18},T_{19},T_{22},T_{16},T_{17},T_{21},T_{20},T_{23} => P_{29},P_{29},P_{30}
95. T_{15},T_{18},T_{19},T_{21},T_{16},T_{17},T_{22},T_{20},T_{23} => P_{29},P_{29},P_{30}
96. T_{15},T_{18},T_{19},T_{22},T_{16},T_{17},T_{20},T_{21},T_{23} => P_{29},P_{29},P_{30}
97. T_{15},T_{18},T_{19},T_{20},T_{16},T_{21},T_{17},T_{22},T_{23} => P_{29},P_{29},P_{30}
98. T_{15},T_{18},T_{19},T_{20},T_{16},T_{22},T_{17},T_{21},T_{23} => P_{29},P_{29},P_{30}
99. T_{15},T_{18},T_{19},T_{21},T_{16},T_{20},T_{17},T_{22},T_{23} => P_{29},P_{29},P_{30}
100. T_{15},T_{18},T_{19},T_{22},T_{16},T_{21},T_{17},T_{20},T_{23} => P_{29},P_{29},P_{30}
101. T_{15},T_{18},T_{19},T_{21},T_{16},T_{22},T_{17},T_{20},T_{23} => P_{29},P_{29},P_{30}
102. T_{15},T_{18},T_{19},T_{22},T_{16},T_{20},T_{17},T_{21},T_{23} => P_{29},P_{29},P_{30}
103. T_{15},T_{18},T_{19},T_{20},T_{16},T_{21},T_{22},T_{17},T_{23} => P_{29},P_{29},P_{30}
104. T_{15},T_{18},T_{19},T_{20},T_{16},T_{22},T_{21},T_{17},T_{23} => P_{29},P_{29},P_{30}
105. T_{15},T_{18},T_{19},T_{21},T_{16},T_{20},T_{22},T_{17},T_{23} => P_{29},P_{29},P_{30}
106. T_{15},T_{18},T_{19},T_{22},T_{16},T_{21},T_{20},T_{17},T_{23} => P_{29},P_{29},P_{30}

107. T_{15},T_{18},T_{19},T_{21},T_{16},T_{22},T_{20},T_{17},T_{23} => P_{29},P_{29},P_{30}
108. T_{15},T_{18},T_{19},T_{22},T_{16},T_{20},T_{21},T_{17},T_{23} => P_{29},P_{29},P_{30}
109. T_{15},T_{18},T_{19},T_{20},T_{21},T_{16},T_{17},T_{22},T_{23} => P_{29},P_{29},P_{30}
110. T_{15},T_{18},T_{19},T_{20},T_{22},T_{16},T_{17},T_{21},T_{23} => P_{29},P_{29},P_{30}
111. T_{15},T_{18},T_{19},T_{21},T_{20},T_{16},T_{17},T_{22},T_{23} => P_{29},P_{29},P_{30}
112. T_{15},T_{18},T_{19},T_{22},T_{21},T_{16},T_{17},T_{20},T_{23} => P_{29},P_{29},P_{30}
113. T_{15},T_{18},T_{19},T_{21},T_{22},T_{16},T_{17},T_{20},T_{23} => P_{29},P_{29},P_{30}
114. T_{15},T_{18},T_{19},T_{22},T_{20},T_{16},T_{17},T_{21},T_{23} => P_{29},P_{29},P_{30}
115. T_{15},T_{18},T_{19},T_{20},T_{21},T_{16},T_{22},T_{17},T_{23} => P_{29},P_{29},P_{30}
116. T_{15},T_{18},T_{19},T_{20},T_{22},T_{16},T_{21},T_{17},T_{23} => P_{29},P_{29},P_{30}
117. T_{15},T_{18},T_{19},T_{21},T_{20},T_{16},T_{22},T_{17},T_{23} => P_{29},P_{29},P_{30}
118. T_{15},T_{18},T_{19},T_{22},T_{21},T_{16},T_{20},T_{17},T_{23} => P_{29},P_{29},P_{30}
119. T_{15},T_{18},T_{19},T_{21},T_{22},T_{16},T_{20},T_{17},T_{23} => P_{29},P_{29},P_{30}
120. T_{15},T_{18},T_{19},T_{22},T_{20},T_{16},T_{21},T_{17},T_{23} => P_{29},P_{29},P_{30}
121. T_{15},T_{18},T_{19},T_{20},T_{21},T_{22},T_{16},T_{17},T_{23} => P_{29},P_{29},P_{30}
122. T_{15},T_{18},T_{19},T_{20},T_{22},T_{21},T_{16},T_{17},T_{23} => P_{29},P_{29},P_{30}
123. T_{15},T_{18},T_{19},T_{21},T_{20},T_{22},T_{16},T_{17},T_{23} => P_{29},P_{29},P_{30}
124. T_{15},T_{18},T_{19},T_{22},T_{21},T_{20},T_{16},T_{17},T_{23} => P_{29},P_{29},P_{30}
125. T_{15},T_{18},T_{19},T_{21},T_{22},T_{20},T_{16},T_{17},T_{23} => P_{29},P_{29},P_{30}
126. T_{15},T_{18},T_{19},T_{22},T_{20},T_{21},T_{16},T_{17},T_{23} => P_{29},P_{29},P_{30}

De acordo com as regras da propriedade *soudness*, quando uma ficha aparece no lugar de término (P_{30} para esta rede), todos os outros lugares deveriam estar vazios. Claramente por meio da simulação pode-se comprovar fichas excedentes (P_{29} e P_{29}) na *WorkFlow net* representada pela Figura 5.2, desta forma provando não conformidade com a propriedade *soudness*.

5.1.2 Grafos de alcançabilidade

Uma outra possibilidade de verificação da propriedade *soudness* é usando o grafo de alcançabilidade da rede de Petri correspondente. Os pontos a serem verificados são então:

- Verificar se o grafo possui apenas um estado final;
- Se o estado inicial possui apenas uma marca, o grafo deve apresentar apenas uma marca no estado final;

- Para cada atividade, verificar se existe uma mudança de estado no grafo que corresponda ao disparo da atividade correspondente.

O grafo de alcançabilidade cresce de forma exponencial em tamanho de lugares e transições. Devido a isto, as técnicas de análise são também exponenciais em tempo.

	P20	P21	P22	P23	P24	P25	P26	P27	P28	P29	P30
S0	1	0	0	0	0	0	0	0	0	0	0
S1	0	1	0	0	1	0	0	0	0	0	0
S2	0	1	0	0	0	1	0	0	0	0	0
S3	0	0	1	0	1	0	0	0	0	0	0
S4	0	1	0	0	0	0	1	1	1	0	0
S5	0	0	1	0	0	1	0	0	0	0	0
S6	0	0	0	1	1	0	0	0	0	0	0
S7	0	1	0	0	0	0	1	1	0	1	0
S8	0	1	0	0	0	0	1	0	1	1	0
S9	0	1	0	0	0	0	0	1	1	1	0
S10	0	0	1	0	0	0	1	1	1	0	0
S11	0	0	0	1	0	1	0	0	0	0	0
S12	0	1	0	0	0	0	1	0	0	2	0
S13	0	1	0	0	0	0	0	1	0	2	0
S14	0	0	1	0	0	0	1	1	0	1	0
S15	0	1	0	0	0	0	0	0	1	2	0
S16	0	0	1	0	0	0	1	0	1	1	0
S17	0	0	1	0	0	0	0	1	1	1	0
S18	0	0	0	1	0	0	1	1	1	0	0
S19	0	1	0	0	0	0	0	0	0	3	0
S20	0	0	1	0	0	0	1	0	0	2	0
S21	0	0	1	0	0	0	0	1	0	2	0
S22	0	0	0	1	0	0	1	1	0	1	0
S23	0	0	1	0	0	0	0	0	1	2	0
S24	0	0	0	1	0	0	1	0	1	1	0
S25	0	0	0	1	0	0	0	1	1	1	0
S26	0	0	1	0	0	0	0	0	0	3	0
S27	0	0	0	1	0	0	1	0	0	2	0
S28	0	0	0	1	0	0	0	1	0	2	0
S29	0	0	0	0	0	0	1	1	0	0	1
S30	0	0	0	1	0	0	0	0	1	2	0
S31	0	0	0	0	0	0	1	0	1	0	1
S32	0	0	0	0	0	0	0	1	1	0	1
S33	0	0	0	1	0	0	0	0	0	3	0
S34	0	0	0	0	0	0	1	0	0	1	1
S35	0	0	0	0	0	0	0	1	0	1	1
S36	0	0	0	0	0	0	0	0	1	1	1

Tabela 5.1: Tabela representando marcações do grafo

A Tabela 5.1 obtida usando um dos módulos do PIPE, representa as marcações referentes aos estados do grafo de alcançabilidade apresentando na Figura 5.3 gerada a partir da *WorkFlow net* representada na Figura 5.2. Fica evidente que a *WorkFlow net* não é *sound* pelo falto de seu grafo de alcançabilidade possuir mais de um estado final.

5.1.3 Verificação da propriedade *Soundness* usando a Lógica Linear nos Modelos sem Recursos

A árvore de prova da lógica linear formaliza a propriedade *soundness* e determina, em particular, as restrições de causalidade que exitem na *quest* analisada. Uma das grandes vantagens da lógica linear é permitir estudar etapas de processos da *WorkFlow*,

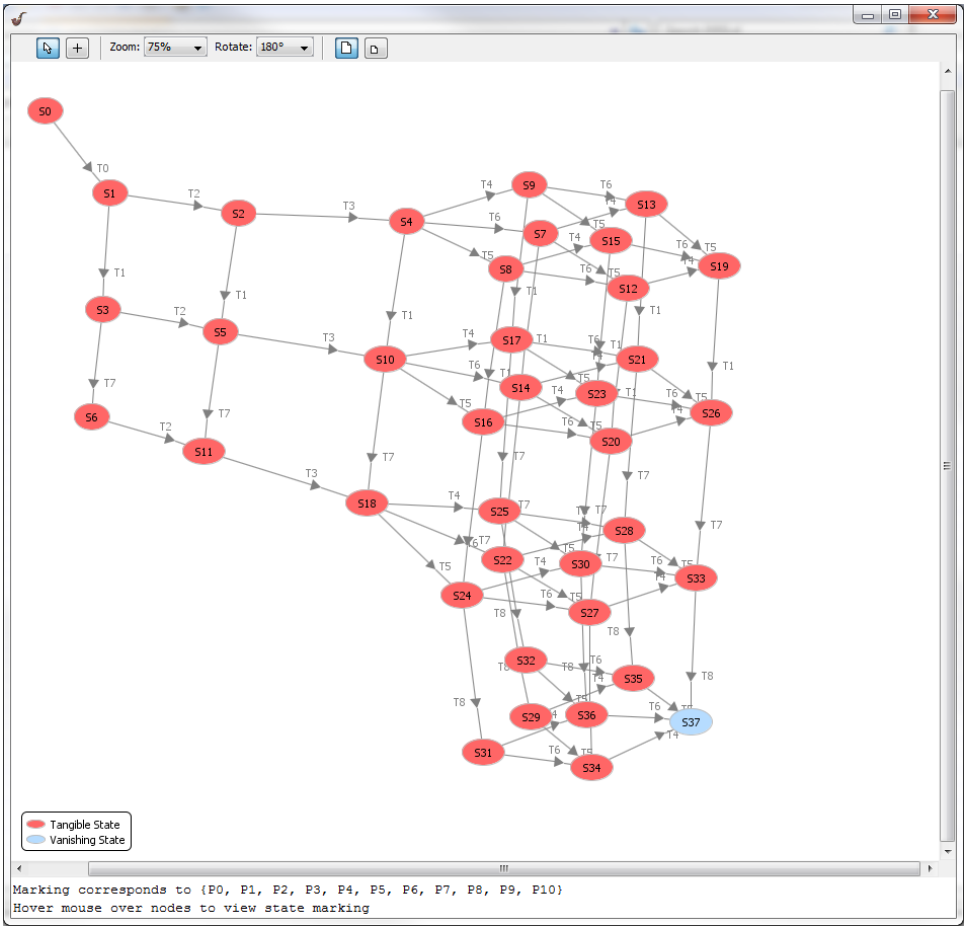

Figura 5.3: Grafo das marcações

considerando marcações parciais da rede de Petri correspondentes (apenas as fichas diretamente envolvidas no processo que será analisado). De fato, a lógica linear [Couvreur e Poitrenaud 2009] permite o estudo do problema de alcançabilidade das redes de Petri considerando somente uma marcação parcial da rede em vez da marcação completa, como é o caso das técnicas baseadas em grafos de marcações acessíveis.

Um jogo será composto de várias *quests*, cada uma delas representando um sub processo de *WorkFlow*. Assim, a abordagem proposta nessa dissertação autoriza o estudo separado das diversas *quests* e é por meio da integração das diversas *quests* na rede de *quests*, que o resultado completo da análise do jogo em particular sera estabelecido. Se acontecer uma mudança em uma das *quests* estudadas, um novo estudo das boas propriedades será realizado somente para a *quest* que sofreu a modificação. Considerando abordagens baseadas em grafo de alcançabilidade, se alguma mudança acontece numa *quest* um novo grafo de alcançabilidade deverá ser reconstruído para o modelo do jogo completo.

Mesmo que a complexidade subjacente da abordagem proposta parece equivalente em complexidade com a construção do grafo de alcançabilidade, a verificação da propriedade *soundness* será realizada de forma gradativa considerando cada *quest* de forma distinta no processo de análise.

Considerando uma *quest* específica representada para uma *WorkFlow net*, é necessário para provar a propriedade de *soundness* representar por fórmulas da lógica linear o modelo correspondente. Se existem roteiros seletivos na *WorkFlow net*, mais de um sequente da lógica linear deverá ser provado.

Um cenário no contexto das *WorkFlow nets* corresponde a uma rota específica no processo e se a *WorkFlow net* tem mais de uma rota (lugares com mais de uma transição de saída) é então necessário construir um sequente por cada alternativa proposta. Por exemplo, na *quest* da Figura 5.4 existem dois cenários diferentes: o primeiro Sc_1 onde todas as atividades da *quest* incluindo a optativa(find gacha seed t_3) serão executadas e o segundo Sc_2, onde a atividade optativa não será considerada.

É importante entender na abordagem proposta a verdadeira natureza dos cenários. Na lógica linear um cenário é um conjunto de disparo de transições produzidos a partir de uma sequência de disparos parcialmente ordenada que a partir da marcação inicial vai produzir uma marcação final. Por exemplo, o cenário Sc_1 da *quest* número 6 poderia ser definido indiferentemente pelo sequente $Start, T_0, T_1, T_2, T_3, T_4, T_5, T_6, T_7, T_8, T_9, T_{10}, T_{11}, T_{12}, T_{14} \vdash End$ ou pelo sequente $Start, T_0, T_1, T_2, T_3, T_5, T_4, T_6, T_7, T_8, T_9, T_{10}, T_{11}, T_{12}, T_{14} \vdash End$ já que o meta-conectivo "," do sequente é comutativo.

No caso de um grafo de alcançabilidade, um cenário será um conjunto de transições completamente ordenado. Consequentemente, quando existir paralelismo na *quest*, um cenário definido por um sequente da lógica linear corresponde na verdade a um conjunto de cenários no grafo por causa das situações de paralelismo. Em particular, se duas atividades A e B acontecem em paralelo, o grafo de alcançabilidade irá produzir duas sequências possíveis, A seguido de B ou B seguido de A, mesmo não existindo uma relação de causalidade entre as atividades A e B. A complexidade adicional produzida pelas estruturas paralelas das redes de Petri desaparecem na árvore de prova da lógica linear e a formalização da propriedade de *soundness* representará a verdadeira estrutura da *quest* dada pela *WorkFlow net*. No pior dos casos, a complexidade da abordagem proposta será eventualmente exponencial em função do número de roteiros seletivos da *WorkFlow net* mas os roteiros em paralelos não acrescentarão a complexidade que os métodos baseados em grafos de alcançabilidade produzem.

Após definir os sequentes que representam uma *quest*, esses devem ser provados. Para provar os quentes, árvores de prova serão produzidas. Em particular, cada sequente terá somente um átomo *start* que representa o início da *quest*

Para ilustrar o método de análise proposto nesse trabalho, a *quest* número 7 representada na Figura 5.2, será considerada. O seguinte sequente precisa então ser provado:

$$P_{20}, T_{15}, T_{16}, T_{17}, T_{18}, T_{19}, T_{20}, T_{21}, T_{22}, T_{23} \vdash P_{30}$$

onde:

$T_{15} = P_{20} \multimap P_{21} \otimes P_{24}$

$T_{16} = P_{21} \multimap P_{22}$

$T_{17} = P_{22} \multimap P_{23}$

$T_{18} = P_{24} \multimap P_{25}$

$T_{19} = P_{25} \multimap P_{26} \otimes P_{27} \otimes P_{28}$

$T_{20} = P_{26} \multimap P_{29}$

$T_{21} = P_{27} \multimap P_{29}$

$T_{22} = P_{28} \multimap P_{29}$

$T_{23} = P_{29} \otimes P_{23} \multimap P_{30}$

Aplicando as regras do cálculo de sequentes a este sequente linear, é possível provar se o mesmo é ou não um sequente sintaticamente válido. A árvore de prova realiza a verificação da propriedade *soundness* da *quest* representada pela Figura 5.2.

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{P_{23}\vdash P_{23} \quad P_{29}\vdash P_{29}}{P_{23},P_{29}\vdash P_{23}\otimes P_{29}}\otimes_R \quad P_{27},P_{28},T_{21},T_{22},P_{30}\vdash P_{30}}{P_{26}\vdash P_{26} \quad P_{23},P_{27},P_{28},P_{29},T_{21},T_{22},P_{23}\otimes P_{29}\multimap P_{30}\vdash P_{30}}\multimap_L}{P_{23},P_{26},P_{27},P_{28},P_{26}\multimap P_{29},T_{21},T_{22},T_{23}\vdash P_{30}}\otimes_L}{P_{23},P_{26},P_{27}\otimes P_{28},T_{20},T_{21},T_{22},T_{23}\vdash P_{30}}\otimes_L}{P_{25}\vdash P_{25} \quad P_{23},P_{26}\otimes P_{27}\otimes P_{28},T_{20},T_{21},T_{22},T_{23}\vdash P_{30}}\multimap_L}{P_{24}\vdash P_{24} \quad P_{23},P_{25},P_{25}\multimap P_{26}\otimes P_{27}\otimes P_{28},T_{20},T_{21},T_{22},T_{23}\vdash P_{30}}\multimap_L}{P_{22}\vdash P_{22} \quad P_{24},P_{23},P_{24}\multimap P_{25},T_{19},T_{20},T_{21},T_{22},T_{23}\vdash P_{30}}\multimap_L}{P_{21}\vdash P_{21} \quad P_{24},P_{22},P_{22}\multimap P_{23},T_{18},T_{19},T_{20},T_{21},T_{22},T_{23}\vdash P_{30}}\multimap_L}{P_{21},P_{24},P_{21}\multimap P_{22},T_{17},T_{18},T_{19},T_{20},T_{21},T_{22},T_{23}\vdash P_{30}}\otimes_L}{P_{20}\vdash P_{20} \quad P_{21}\otimes P_{24},P_{21}\multimap P_{22},T_{17},T_{18},T_{19},T_{20},T_{21},T_{22},T_{23}\vdash P_{30}}\multimap_L}{P_{20},P_{20}\multimap P_{21}\otimes P_{24},T_{16},T_{17},T_{18},T_{19},T_{20},T_{21},T_{22},T_{23}\vdash P_{30}}$$

O resultado da prova mostra que a *quest* não é *sound* porque não existem mais regras a serem aplicadas para o sequênte.

5.1.4 Verificação da propriedade *Soundness* usando a Lógica Linear nos Modelo com Recursos

Durante uma *quest* de um jogo, alguns itens podem ser encontrados. Tais itens podem se tornar mais tarde necessários à realização de outras *quests* e serão vistos na *WorkFlow net* correspondente como recursos discretos que poderão ser produzidos e consumidos.

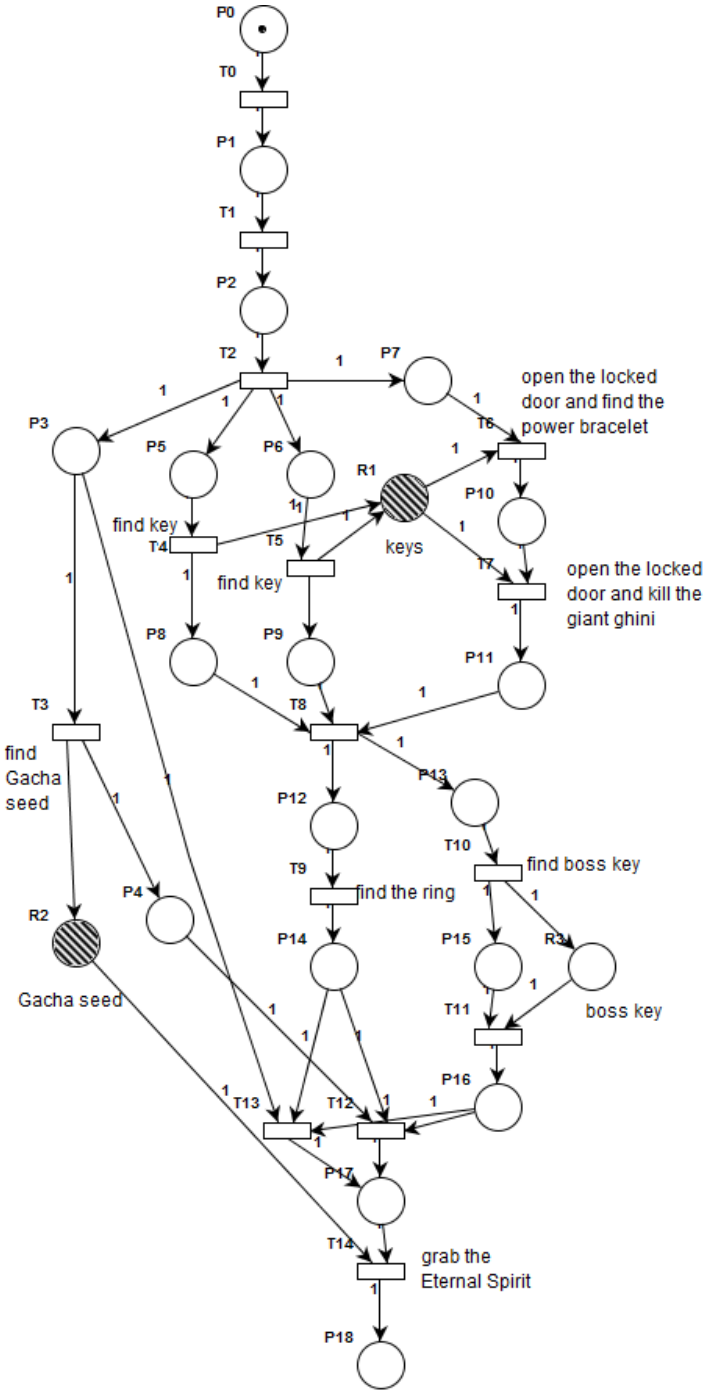

Figura 5.4: Quest com recursos não sound

Por exemplo, na *quest* representada pela Figura 5.4 o recurso R_2 terá de ser repassado

para a próxima *quest*; sendo assim, o mesmo deverá estar conectado à ultima transição T_{14} que finaliza a *quest*. Nesta *quest*, existem dois cenários: o primeiro cenário Sc_1 onde o recurso R_2 é produzido, e o segundo cenário Sc_2 onde o recurso R_2 não é produzido. Assim, para a *WorkFlow net* da Figura 5.4, dois sequentes da lógica linear deverão ser provados.

Para o cenário Sc_1, é necessário provar o sequente:

$$Start, T_0, T_1, T_2, T_3, T_4, T_5, T_6, T_7, T_8, T_9, T_{10}, T_{11}, T_{12}, T_{14} \vdash End$$

e para o cenário SC_2 é necessário provar o sequente:

$$Start, T_0, T_1, T_2, T_4, T_5, T_6, T_7, T_8, T_9, T_{10}, T_{11}, T_{13}, T_{14} \vdash End$$

onde: $Start = P_0$ e $End = P_{18}$.

Para provar os sequentes dos cenários SC_1 e SC_2, árvores de prova da lógica linear são então produzidas.

Árvore de prova para o cenário SC_1:

$$\underline{\dfrac{P_{17}\vdash P_{17} \qquad R_2\vdash R_2}{P_{17},R_2\vdash P_{17}\otimes R_2}\otimes_R \qquad P_{18}\vdash P_{18}}\ \multimap_L$$

$$\underline{\dfrac{P_4\vdash P_4 \qquad P_{14}\vdash P_{14} \qquad P_{16}\vdash P_{16}}{P_4,P_{14},P_{16}\vdash P_4\otimes P_{14}\otimes P_{16}}\otimes_R \qquad R_2,P_{17},P_{17}\otimes R_2\multimap P_{18}\vdash P_{18}}\ \multimap_L$$

$$\underline{P_3\vdash P_3 \qquad P_4,P_{14},P_{16},R_2,P_4\otimes P_{14}\otimes P_{16}\multimap P_{17},T_{14}\vdash P_{18}}\ \otimes_L$$

$$\underline{P_3\vdash P_3 \qquad P_{14},P_{16},P_4\otimes R_2,P_4\otimes P_{14}\otimes P_{16}\multimap P_{17},T_{14}\vdash P_{18}}\ \multimap_L$$

$$\underline{P_{15}\otimes R_3\vdash P_{15}\otimes R_3 \qquad P_3,P_{14},P_{16},P_3\multimap P_4\otimes R_2,T_{12},T_{14}\vdash P_{18}}\ \multimap_L$$

$$\underline{P_{13}\vdash P_{13} \qquad P_3,P_{14},P_{15}\otimes R_3,P_{15}\otimes R_3\multimap P_{16},T_3,T_{12},T_{14}\vdash P_{18}}\ \multimap_L$$

$$\underline{P_{12}\vdash P_{12} \qquad P_3,P_{13},P_{14},P_{13}\multimap P_{15}\otimes R_3,T_3,T_{11},T_{12},T_{14}\vdash P_{18}}\ \multimap_L$$

$$\underline{P_3,P_{12},P_{13},P_{12}\multimap P_{14},T_3,T_{10},T_{11},T_{12},T_{14}\vdash P_{18}}\ \otimes_L$$

$$\underline{\dfrac{P_8\vdash P_8 \qquad P_9\vdash P_9 \qquad P_{11}\vdash P_{11}}{P_8,P_9,P_{11}\vdash P_8\otimes P_9\otimes P_{11}}\otimes_R \qquad P_3,P_{12}\otimes P_{13},P_{12}\multimap P_{14},T_3,T_{10},T_{11},T_{12},T_{14}\vdash P_{18}}\ \multimap_L$$

$$\underline{\dfrac{P_{10}\vdash P_{10} \qquad R_1\vdash R_1}{P_{10},R_1\vdash P_{10}\otimes R_1}\otimes_R \qquad P_3,P_8,P_9,P_{11},P_8\otimes P_9\otimes P_{11}\multimap P_{12}\otimes P_{13},T_3,T_9,T_{10},T_{11},T_{12},T_{14}\vdash P_{18}}\ \multimap_L$$

$$\underline{\dfrac{P_7\vdash P_7 \qquad R_1\vdash R_1}{P_7,R_1\vdash P_7\otimes R_1}\otimes_R \qquad P_3,P_8,R_1,P_9,P_{10},P_{10}\otimes R_1\multimap P_{11},T_3,T_8,T_9,T_{10},T_{11},T_{12},T_{14}\vdash P_{18}}\ \multimap_L$$

$$\underline{P_3,P_7,P_8,R_1,P_9,R_1,P_7\otimes R_1\multimap P_{10},T_3,T_7,T_8,T_9,T_{10},T_{11},T_{12},T_{14}\vdash P_{18}}\ \otimes_L$$

$$\underline{P_3,P_7,P_8,R_1,P_9\otimes R_1,P_7\otimes R_1\multimap P_{10},T_3,T_7,T_8,T_9,T_{10},T_{11},T_{12},T_{14}\vdash P_{18}}\ \otimes_L$$

$$\underline{P_6\vdash P_6 \qquad P_3,P_7,P_8\otimes R_1,P_9\otimes R_1,P_7\otimes R_1\multimap P_{10},T_3,T_7,T_8,T_9,T_{10},T_{11},T_{12},T_{14}\vdash P_{18}}\ \multimap_L$$

$$\underline{P_5\vdash P_5 \qquad P_3,P_6,P_7,P_8\otimes R_1,P_6\multimap P_9\otimes R_1,T_3,T_6,T_7,T_8,T_9,T_{10},T_{11},T_{12},T_{14}\vdash P_{18}}\ \multimap_L$$

$$\underline{P_3,P_5,P_6,P_7,P_5\multimap P_8\otimes R_1,T_3,T_5,T_6,T_7,T_8,T_9,T_{10},T_{11},T_{12},T_{14}\vdash P_{18}}\ \otimes_L$$

$$\underline{P_3,P_5,P_6\otimes P_7,P_5\multimap P_8\otimes R_1,T_3,T_5,T_6,T_7,T_8,T_9,T_{10},T_{11},T_{12},T_{14}\vdash P_{18}}\ \otimes_L$$

$$\underline{P_3,P_5\otimes P_6\otimes P_7,P_5\multimap P_8\otimes R_1,T_3,T_5,T_6,T_7,T_8,T_9,T_{10},T_{11},T_{12},T_{14}\vdash P_{18}}\ \otimes_L$$

$$\underline{P_2\vdash P_2 \qquad P_3\otimes P_5\otimes P_6\otimes P_7,P_5\multimap P_8\otimes R_1,T_3,T_5,T_6,T_7,T_8,T_9,T_{10},T_{11},T_{12},T_{14}\vdash P_{18}}\ \multimap_L$$

$$\underline{P_1\vdash P_1 \qquad P_2,P_2\multimap P_3\otimes P_5\otimes P_6\otimes P_7,T_3,T_4,T_5,T_6,T_7,T_8,T_9,T_{10},T_{11},T_{12},T_{14}\vdash P_{18}}\ \multimap_L$$

$$\underline{P_0\vdash P_0 \qquad P_1,P_1\multimap P_2,T_2,T_3,T_4,T_5,T_6,T_7,T_8,T_9,T_{10},T_{11},T_{12},T_{14}\vdash P_{18}}\ \multimap_L$$

$$P_0,P_0\multimap P_1,T_1,T_2,T_3,T_4,T_5,T_6,T_7,T_8,T_9,T_{10},T_{11},T_{12},T_{14}\vdash P_{18}$$

Árvore de prova para o cenário SC_2:

$$\dfrac{P_3\vdash P_3 \qquad P_{14}\vdash P_{14} \qquad P_{16}\vdash P_{16}}{P_3,P_{14},P_{16}\vdash P_3\otimes P_{14}\otimes P_{16}}\ \otimes_R \qquad P_{17},P_{17}\otimes R_2\multimap P_{18}\vdash P_{18} \quad \multimap_L$$

$$P_{15}\otimes R_3\vdash P_{15}\otimes R_3 \qquad P_3,P_{14},P_{16},P_3\otimes P_{14}\otimes P_{16}\multimap P_{17},T_{14}\vdash P_{18} \quad \multimap_L$$

$$P_{13}\vdash P_{13} \qquad P_3,P_{14},P_{15}\otimes R_3,P_{15}\otimes R_3\multimap P_{16},T_{13},T_{14}\vdash P_{18} \quad \multimap_L$$

$$P_{12}\vdash P_{12} \qquad P_3,P_{13},P_{14},P_{13}\multimap P_{15}\otimes R_3,T_{11},T_{13},T_{14}\vdash P_{18} \quad \multimap_L$$

$$P_3,P_{12},P_{13},P_{12}\multimap P_{14},T_{10},T_{11},T_{13},T_{14}\vdash P_{18} \quad \otimes_L$$

$$\dfrac{P_8\vdash P_8 \qquad P_9\vdash P_9 \qquad P_{11}\vdash P_{11}}{P_8,P_9,P_{11}\vdash P_8\otimes P_9\otimes P_{11}}\ \otimes_R \qquad P_3,P_{12}\otimes P_{13},P_{12}\multimap P_{14},T_{10},T_{11},T_{13},T_{14}\vdash P_{18} \quad \multimap_L$$

$$\dfrac{P_{10}\vdash P_{10} \qquad R_1\vdash R_1}{P_{10},R_1\vdash P_{10}\otimes R_1}\ \otimes_R \qquad P_3,P_8,P_9,P_{11},P_8\otimes P_9\otimes P_{11}\multimap P_{12}\otimes P_{13},T_9,T_{10},T_{11},T_{13},T_{14}\vdash P_{18} \quad \multimap_L$$

$$\dfrac{P_7\vdash P_7 \qquad R_1\vdash R_1}{P_7,R_1\vdash P_7\otimes R_1}\ \otimes_R \qquad P_3,P_8,R_1,P_9,P_{10},P_{10}\otimes R_1\multimap P_{11},T_8,T_9,T_{10},T_{11},T_{13},T_{14}\vdash P_{18} \quad \multimap_L$$

$$P_3,P_7,P_8,R_1,P_9,R_1,P_7\otimes R_1\multimap P_{10},T_7,T_8,T_9,T_{10},T_{11},T_{13},T_{14}\vdash P_{18} \quad \otimes_L$$

$$P_3,P_7,P_8,R_1,P_9\otimes R_1,P_7\otimes R_1\multimap P_{10},T_7,T_8,T_9,T_{10},T_{11},T_{13},T_{14}\vdash P_{18} \quad \otimes_L$$

$$P_6\vdash P_6 \qquad P_3,P_7,P_8\otimes R_1,P_9\otimes R_1,P_7\otimes R_1\multimap P_{10},T_7,T_8,T_9,T_{10},T_{11},T_{13},T_{14}\vdash P_{18} \quad \multimap_L$$

$$P_5\vdash P_5 \qquad P_3,P_6,P_7,P_8\otimes R_1,P_6\multimap P_9\otimes R_1,T_6,T_7,T_8,T_9,T_{10},T_{11},T_{13},T_{14}\vdash P_{18} \quad \multimap_L$$

$$P_3,P_5,P_6,P_7,P_5\multimap P_8\otimes R_1,T_5,T_6,T_7,T_8,T_9,T_{10},T_{11},T_{13},T_{14}\vdash P_{18} \quad \otimes_L$$

$$P_3,P_5,P_6\otimes P_7,P_5\multimap P_8\otimes R_1,T_5,T_6,T_7,T_8,T_9,T_{10},T_{11},T_{13},T_{14}\vdash P_{18} \quad \otimes_L$$

$$P_3,P_5\otimes P_6\otimes P_7,P_5\multimap P_8\otimes R_1,T_5,T_6,T_7,T_8,T_9,T_{10},T_{11},T_{13},T_{14}\vdash P_{18} \quad \otimes_L$$

$$P_2\vdash P_2 \qquad P_3\otimes P_5\otimes P_6\otimes P_7,P_5\multimap P_8\otimes R_1,T_5,T_6,T_7,T_8,T_9,T_{10},T_{11},T_{13},T_{14}\vdash P_{18} \quad \multimap_L$$

$$P_1\vdash P_1 \qquad P_2,P_2\multimap P_3\otimes P_5\otimes P_6\otimes P_7,T_4,T_5,T_6,T_7,T_8,T_9,T_{10},T_{11},T_{13},T_{14}\vdash P_{18} \quad \multimap_L$$

$$P_0\vdash P_0 \qquad P_1,P_1\multimap P_2,T_2,T_4,T_5,T_6,T_7,T_8,T_9,T_{10},T_{11},T_{13},T_{14}\vdash P_{18} \quad \multimap_L$$

$$P_0,P_0\multimap P_1,T_1,T_2,T_4,T_5,T_6,T_7,T_8,T_9,T_{10},T_{11},T_{13},T_{14}\vdash P_{18}$$

No final da prova do cenário Sc_2 nenhum recurso é produzido em R_2 como indicado no final da árvore de prova $P_{17},P_{17}\otimes R_2\multimap P_{18}\vdash P_{18}$, e consequentemente a *WorkFlow net* correspondente não é *sound* já que nenhuma regra pode ser aplicada no sequente para que uma ficha seja produzida no lugar *End*. Isso mostra até um certo ponto que a atividade associada à transição T_3 ("find the gacha seed") deveria ser obrigatória para poder realizar a próxima *quest* número 7.

5.1.5 Verificação da propriedade Soundness numa Rede de Quest usando a lógica linear

Uma rede de *quest* consiste de uma *WorkFlow net* que representa as interligações entre as *quests* de um mesmo nível, ou simplesmente de um jogo. É evidente que a introdução de itens de jogo necessários à boa realização das *quests* podem introduzir novas informações

sobre a factibilidade do modelo da rede de *quest*. Uma rede de *quest* sem recursos pode ser perfeitamente *sound* enquanto a mesma rede com a representação dos itens de jogo pode se tornar não *sound*.

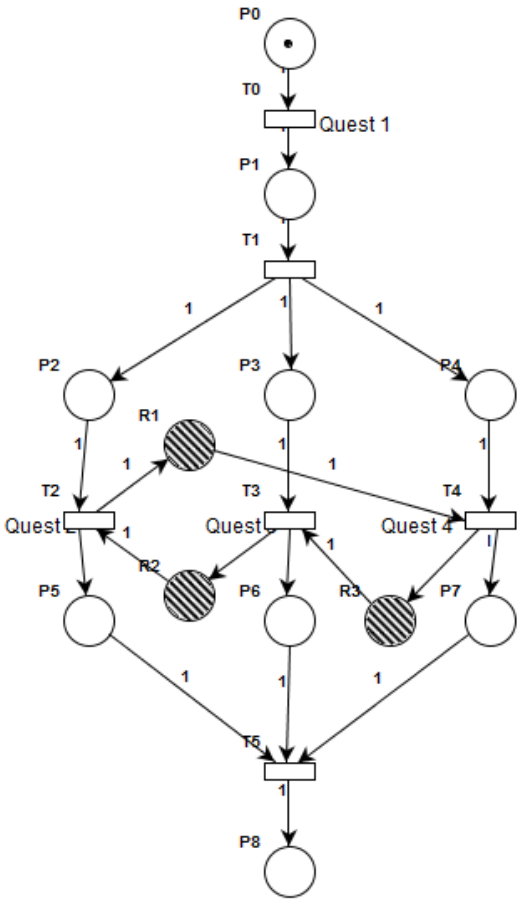

Figura 5.5: Rede de Quests com recursos não sound

Para ilustrar tal situação considera-se o exemplo da Figura 5.5. Considerando o cálculo do sequente sem os recursos, o resultado é positivo (rede *sound*), pois como pode ser visto após a aplicação das regras de dedução, o sequente é comprovado.

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{P_5 \vdash P_5 \quad P_6 \vdash P_6 \quad P_7 \vdash P_7}{P_5, P_6, P_7 \vdash P_5 \otimes P_6 \otimes P_7}\otimes_R \quad P_8 \vdash P_8}{P_4 \vdash P_4 \quad P_5, P_6, P_7, P_5 \otimes P_6 \otimes P_7 \multimap P_8 \vdash P_8}\multimap_L}{P_3 \vdash P_3 \quad P_4, P_5, P_6, P_4 \multimap P_7, T_5 \vdash P_8}\multimap_L}{P_2 \vdash P_2 \quad P_3, P_4, P_5, P_3 \multimap P_6, T_4, T_5 \vdash P_8}\multimap_L}{P_2, P_3, P_4, P_2 \multimap P_5, T_3, T_4, T_5 \vdash P_8}\otimes_L}{P_2, P_3 \otimes P_4, P_2 \multimap P_5, T_3, T_4, T_5 \vdash P_8}\otimes_L}{P_1 \vdash P_1 \quad P_2 \otimes P_3 \otimes P_4, P_2 \multimap P_5, T_3, T_4, T_5 \vdash P_8}\multimap_L}{P_0 \vdash P_0 \quad P_1, P_1 \multimap P_2 \otimes P_3 \otimes P_4, T_2, T_3, T_4, T_5 \vdash P_8}\multimap_L$$

$$P_0, P_0 \multimap P_1, T_1, T_2, T_3, T_4, T_5 \vdash P_8$$

Porém ao considerar os recursos, como pode ser visto no calculo do sequente:

$$\dfrac{\dfrac{\dfrac{\dfrac{P_2, P_3, P_4, P_2 \otimes R_3 \multimap P_5 \otimes R_1, T_3, T_4, T_5 \vdash P_8}{P_2, P_3 \otimes P_4, P_2 \otimes R_3 \multimap P_5 \otimes R_1, T_3, T_4, T_5 \vdash P_8}\otimes_L}{P_1 \vdash P_1 \quad P_2 \otimes P_3 \otimes P_4, P_2 \otimes R_3 \multimap P_5 \otimes R_1, T_3, T_4, T_5 \vdash P_8}\multimap_L}{P_0 \vdash P_0 \quad P_1, P_1 \multimap P_2 \otimes P_3 \otimes P_4, T_2, T_3, T_4, T_5 \vdash P_8}\multimap_L$$

$$P_0, P_0 \multimap P_1, T_1, T_2, T_3, T_4, T_5 \vdash P_8$$

Após o disparo da transição T_1, teremos fichas nos lugares P_2, P_3 e P_4; no entanto, nenhuma regra da lógica linear pode ser aplicada, assim entrando em situação de *deadlock*.

5.2 Análise Quantitativa de uma Quest

O principal objetivo do planejamento de quests, apresentado neste trabalho é calcular janelas de datas onde serão realizadas as atividades das *quests*. Assim, será possível prever o tempo de jogo das *quests* assim como o tempo de jogo do game completo.

No caso das *WorkFlow nets* analisadas por meio das árvores de prova canônica da lógica linear, as datas de execução das quests poderão ser dadas através de datas simbólicas, ao invés de datas numéricas.

A maior vantagem da utilização de datas simbólicas é que quando estas já estiverem calculadas, poderão ser utilizadas diretamente para qualquer caso a ser tratado pelo processo de *WorkFlow* analisado. Em particular, na fase de *design* de um jogo, dificilmente será possível realizar uma estimativa do tempo das atividades do jogo e somente um tipo de simulação simbólica poderá ser efetuada.

A análise quantitativa também considera todos os possíveis cenários de uma t-Time *WorkFlow net*. Isto é necessário uma vez que a duração de uma *quest* pode variar de acordo com as escolhas do jogador. Por exemplo, se o jogador decide não realizar as *side-quests* o tempo de jogo da *quest* será reduzido.

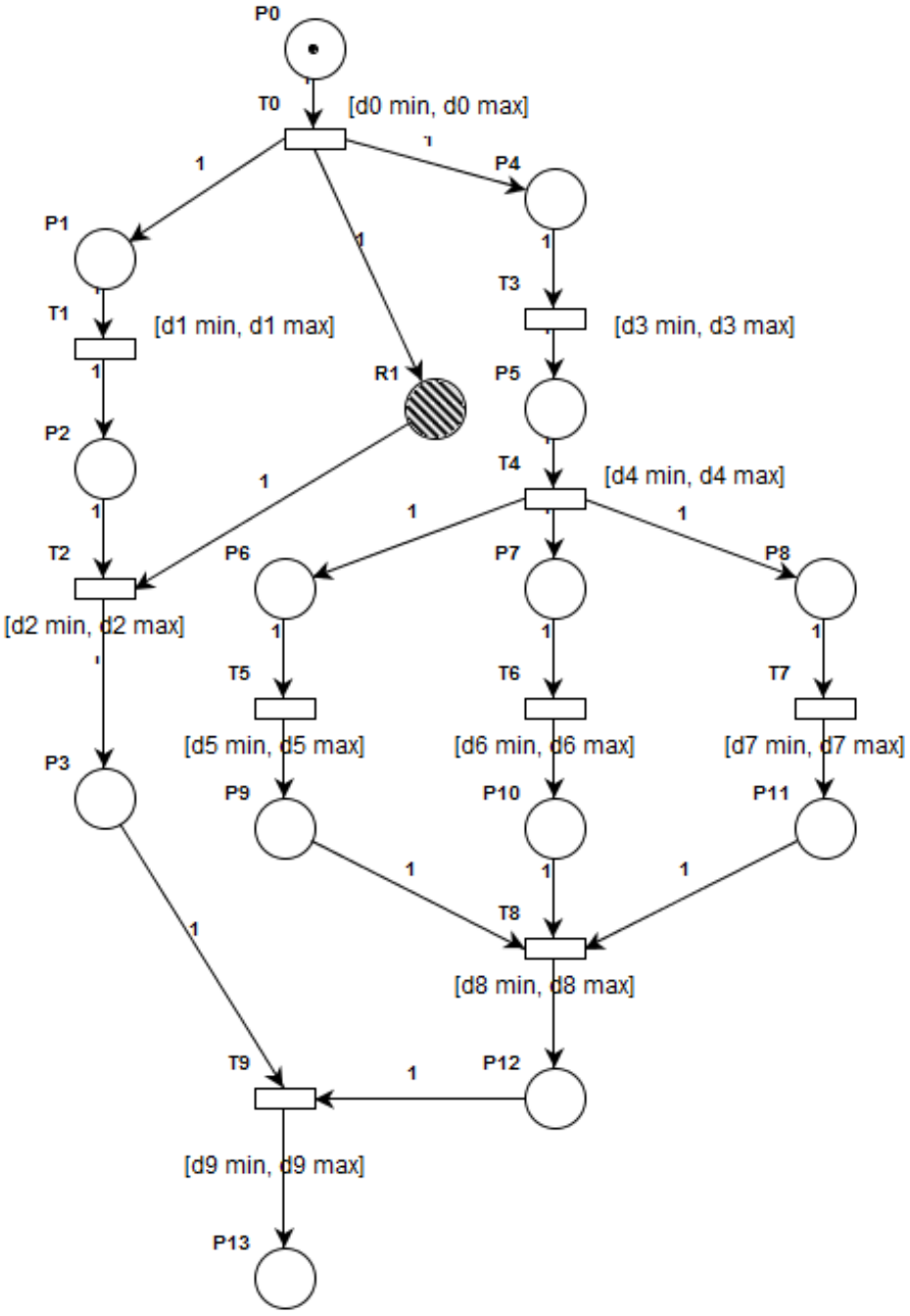

Figura 5.6: t-Time *WorkFlow net* com datas simbólicas da quest 7

Para ilustrar a abordagem proposta, considera-se a t-Time *WorkFlow net* mostrada na Figura 5.6 [Passos 2009]. Considerando as simplificações seguintes:

$$Seq_1 = D_S + d_0 + d_1$$

$$Seq_2 = D_S + d_0 + d_1 + d_2$$

$$Seq_3 = D_S + d_0 + d_3$$

$$Seq_4 = D_S + d_0 + d_3 + d_4$$

$$Seq_5 = D_S + d_0 + d_3 + d_4 + d_5$$

$$Seq_6 = D_S + d_0 + d_3 + d_4 + d_6$$

$$Seq_7 = D_S + d_0 + d_3 + d_4 + d_7$$

$$Seq_8 = Seq_4 + max(d_5, d_6, d_7) + d_8$$

$$Seq_9 = max(Seq_2, Seq_8) + d_9$$

Onde D_S representa a data de início da *quest*, a árvore obtida com as datas simbólicas é a seguinte:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{P_3(Seq_2,Seq_9)\vdash P_3 \quad P_{12}(Seq_8,Seq_9)\vdash P_{12}}{P_3(Seq_2,Seq_9),P_{12}(Seq_8,Seq_9)\vdash P_3\otimes P_{12}}\otimes_R \quad P_{13}(Seq_9,.)\vdash P_{13}}{P_3(Seq_2,.),P_{12}(Seq_8,.),P_3\otimes P_{12}\multimap P_{13}\vdash P_{13}}\multimap_L}{P_8(Seq_4,Seq_7)\vdash P_8 \quad P_3(Seq_2,.),P_9(Seq_5,.),P_{10}(Seq_6,.),P_{11}(Seq_7,.),P_9\otimes P_{10}\otimes P_{11}\multimap P_{12},T_9\vdash P_{13}}\multimap_L}{P_7(Seq_4,Seq_6)\vdash P_7 \quad P_3(Seq_2,.),P_8(Seq_4,.),P_9(Seq_5,.),P_{10}(Seq_6,.),P_8\multimap P_{11},T_8,T_9\vdash P_{13}}\multimap_L}{P_6(Seq_4,Seq_5)\vdash P_6 \quad P_3(Seq_2,.),P_7(Seq_4,.),P_8(Seq_4,.),P_9(Seq_5,.),P_7\multimap P_{10},T_7,T_8,T_9\vdash P_{13}}\multimap_L}{P_3(Seq_2,.),P_6(Seq_4,.),P_7(Seq_4,.),P_8(Seq_4,.),P_6\multimap P_9,T_6,T_7,T_8,T_9\vdash P_{13}}\otimes_L}{P_5(Seq_3,Seq_4)\vdash P_5 \quad P_3(Seq_2,.),P_6(Seq_4,.)\otimes P_7(Seq_4,.)\otimes P_8(Seq_4,.),P_6\multimap P_9,T_6,T_7,T_8,T_9\vdash P_{13}}\multimap_L}{P_4(D_S+d_0,Seq_3)\vdash P_4 \quad P_3(Seq_2,.),P_5(Seq_3,.),P_5\multimap P_6\otimes P_7\otimes P_8,T_5,T_6,T_7,T_8,T_9\vdash P_{13}}\multimap_L}{\dfrac{P_2(Seq_1,Seq_2)\vdash P_2 \quad R_1(D_S+d_0,Seq_2)\vdash R_1}{P_2(Seq_1,Seq_2),R_1(D_S+d_0,Seq_2)\vdash P_2\otimes R_1}\otimes_R \quad P_3(Seq_2,.),P_4(D_S+d_0,.),P_4\multimap P_5,T_4,T_5,T_6,T_7,T_8,T_9\vdash P_{13}}\multimap_L}{P_1(D_S+d_0,Seq_1)\vdash P_1 \quad P_4(D_S+d_0,.),R_1(D_S+d_0,.),P_2(Seq_1,.),P_2\otimes R_1\multimap P_3,T_3,T_4,T_5,T_6,T_7,T_8,T_9\vdash P_{13}}\multimap_L}{P_1(D_S+d_0,.),P_4(D_S+d_0,.),R_1(D_S+d_0,.),P_1\multimap P_2,T_2,T_3,T_4,T_5,T_6,T_7,T_8,T_9\vdash P_{13}}\otimes_L}{P_0(D_S,D_S+d_0)\vdash P_0 \quad P_1(D_S+d_0,.)\otimes P_4(D_S+d_0,.)\otimes R_1(D_S+d_0,.),T_1,T_2,T_3,T_4,T_5,T_6,T_7,T_8,T_9\vdash P_{13}}\multimap_L$$

$$P_0(D_S,.),P_0\multimap P_1\otimes P_4\otimes R_1,T_1,T_2,T_3,T_4,T_5,T_6,T_7,T_8,T_9\vdash P_{13}$$

Para cada atividade elementar modelada na *WorkFlow net* analisada, deve-se extrair as datas de produção, D_P , e consumo, D_C, do átomo que representa a pré-condição da transição correspondente a esta atividade. Quando há mais de uma pré-condição associada à transição, considera-se a data máxima das produções dos átomos correspondentes a estas pré-condições. A data de produção deste átomo, D_P , corresponde ao início da execução

da atividade associada à transição e a data de consumo, DC, corresponde ao término da execução da mesma. Assim, é gerado um intervalo $[D_P, D_C]$ de datas dentro do qual o jogador executará uma referida atividade.

Uma vez que as datas de produção e consumo são dependentes de durações de sensibilização d_i, cujo valor pertence a um intervalo de tempo $\Delta_i = [\delta_{imin}, \delta_{imax}]$, pode-se considerar vários intervalos possíveis de execução das atividades, de acordo com um planejamento estratégico. Por exemplo, o intervalo de execução $I_{Exec} = [D_{Pmin}, D_{Cmax}]$ considera que a alocação do jogador para a execução da atividade poderá ocorrer entre o início ao mais cedo e o término ao mais tarde da atividade considerada. Este intervalo de datas é o mais flexível, no sentido que ele considera a janela de tempo de utilização do jogador a mais extensa possível.

A informação sobre quando o processo global finalizará, para uma dada *quest*, é dado pela data de produção do átomo *End*. Assim, pode-se calcular a data ao mais cedo de finalização do processo, dada pela data mínima D_{Pmin} de produção deste átomo e a data ao mais tarde de finalização do processo, dada pela data máxima D_{Pmax} de produção do átomo *End*.

Transição	**Intervalo de datas**
$T_1 = P_1 \multimap P_2$	$[D_S + d0_{min},$ $D_S + d0_{max} + d1_{max}]$
$T_2 = P_2 \otimes R_1 \multimap P_3$	$[D_S + d0_{min} + d1_{min},$ $D_S + d0_{max} + d1_{max} + d2_{max}]$
$T_3 = P_4 \multimap P_5$	$[D_S + d0_{min},$ $D_S + d0_{max} + d3_{max}]$
$T_4 = P_5 \multimap P_6 \otimes P_7 \otimes P_8$	$[D_S + d0_{min} + d3min,$ $D_S + d0_{max} + d3_{max} + d4_{max}]$
$T_5 = P_6 \multimap P_9$	$[D_S + d0_{min} + d3min + d4_{min},$ $D_S + d0_{max} + d3_{max} + d4_{max} + d5_{max}]$
$T_6 = P_7 \multimap P_{10}$	$[D_S + d0_{min} + d3min + d4_{min},$ $D_S + d0_{max} + d3_{max} + d4_{max} + d6_{max}]$
$T_7 = P_8 \multimap P_{11}$	$[D_S + d0_{min} + d3min + d4_{min},$ $D_S + d0_{max} + d3_{max} + d4_{max} + d7_{max}]$
$T_8 = P_9 \otimes P_{10} \otimes P_{11} \multimap P_{12}$	$[D_S + d0_{min} + d3_{min} +$ $d4_{min} + max(d5_{min}, d6_{min}, d7_{min}),$ $D_S + d0_{max} + d3_{max} + d4_{max} +$ $max(d5_{max}, d6_{max}, d7_{max}) + d8_{max}]$
$T_9 = P_3 \otimes P_{12} \multimap P_{13}$	$[D_S + d0_{min} + max(d1_{min} + d2_{min}, d3_{min} + d4_{min} +$ $max(d5_{min}, d6_{min}, d7_{min}), d8_{min}),$ $D_S + d0_{max} + max(d1_{max} + d2_{max}, d3_{max} + d4_{max} +$ $max(d5_{max}, d6_{max}, d7_{max}) + d8_{m}ax) + d9_{max}]$

Tabela 5.2: Intervalos de datas simbólicas da WorkFlow net t-temporal da Figura 5.6

A Tabela 5.2 mostra os intervalos de datas simbólicas de execução para as ações elementares da *quest*, considerando o intervalo de execução $I_{Exec} = [D_{Pmin}, D_{Cmax}]$. Os

intervalos de datas calculados poderão ser utilizados para planejar a duração do jogo, com base na duração de cada atividade modelada pela t-Time *WorkFlow net* da Figura 5.6.

Se a *quest* começa na data $D_S = 0$ a última atividade realizada representada pela transição T_9 será realizada ao mais cedo na data $d0_{min}+max(d1_{min}+d2_{min}, d3_{min}+d4_{min}+max(d5_{min}, d6_{min}, d7_{min}), d8_{min})$, que representa o tempo mínimo de jogo da *quest*, e ao mais tarde na data $d0_{max}+max(d1_{max}+d2_{max}, d3_{max}+d4_{max}+max(d5_{max}, d6_{max}, d7_{max})+d8_max) + d9_{max}$, que representa o tempo máximo de jogo da *quest*.

Assim, o desenvolvedor do jogo, baseando-se nas informações obtidas, poderá estabelecer tempos razoáveis de durações para as principais atividades, mensurando tempo de jogo suficiente para tornar cada *quest* interessante de acordo com o tipo de jogo (no caso de um "joguinho" de celular as *quests* deverão ser rápidas enquanto que em um jogo de console poderá propor *quests* extensas no tempo).

Capítulo 6

Conclusão e Trabalhos Futuros

Foi apresentado neste trabalho uma proposta de modelagem de jogos de video game baseada nas *WorkFlow nets* e na lógica linear. Na criação do modelo de jogo, além das *WorkFlow nets* tradicionais usadas em sistemas de gerenciamento de *WorkFlow*, foi introduzida a noção de recursos discretos que representam os itens que o jogador encontra no decorrer das *quests* que ele percorre e foi apresentada uma decomposição hierárquica do modelo do jogo com conceito de redes de *quest* que permite agregar o conjunto de *quests* que um jogo completo comporta.

A análise qualitativa apresentada teve por objetivo provar o critério de corretude definido para as *WorkFlow nets* denominado de *soundness*. No contexto de jogos de tipo ação/aventura/RPG.

A análise quantitativa proposta neste trabalho baseou-se no cálculo de datas simbólicas para o planejamento de duração de *quests* de jogos.

As vantagens da abordagem apresentada são diversas. O fato de trabalhar com lógica linear permite provar o critério de corretude soundness em tempo linear, quando se consideram as estruturas paralelas e sem que seja necessária a construção de um grafo das marcações acessíveis, considerando diretamente a própria estrutura da *WorkFlow net* ao invés de considerar seu autômato correspondente.

Além disso, o cálculo de datas simbólicas correspondentes à execução de cada atividade elementar mapeada em uma *t-Time WorkFlow net* permite planejar duração de *quest*, através de fórmulas baseadas em datas simbólicas. O cálculo de datas (ou durações) simbólicas pode ser visto como um tipo de simulação numa etapa do desenvolvimento onde o jogo não foi implementado ainda. O resultado dos dados produzidos poderão em particular orientar na complexidade das atividades propostas para tornar o jogo desafiador do ponto de vista do jogador.

Em relação ao modelo proposto por Liliane Vega [Natkin et al. 2004] que de certa forma foi o ponto de partida desta pesquisa, existem grandes diferenças que devem ser destacadas. Numa rede de transações [Natkin et al. 2004], o conceito de recurso é completamente associado ao jogador. Nas *WorkFlow nets*, o jogador é um caso de um processo

de *WorkFlow* com objetivos específicos a serem atingidos, em particular realizar *quests* produzindo uma ficha no lugar de *End* de uma *WorkFlow net sound*. De fato, a propriedade de *soundness* é específica das *WorkFlow nets* e parece bastante consistente com a noção de *quest* de um jogo de video game. Por outro lado, a noção de recursos discretos neste trabalho foi associada com itens de jogos encontrados e utilizados durante *quests* específicas. Foi mostrado em particular num exemplo que o efeito de tais itens podem facilmente tornar não *sound* uma *quest* ou rede de *quests* que sem a adição destes recursos seria provada como *sound*. Finalmente, os resultados da análise quantitativa permite raciocinar sobre cenários de jogos que ainda não foram implementados, o que constitui uma evolução em relação ao trabalho de Liliane Vega [Natkin et al. 2004] que se baseava exclusivamente em dados de tipo numéricos.

Como trabalhos futuros, diversas propostas podem ser formuladas, em particular:

- Mostrar que uma *quest* de um jogo pode ser formalizada como um bloco bem formado de uma rede de *quests*.
- criar modelos de representação do ambiente do jogo (áreas e objetos existentes no mapa do jogo) e formalizar mecanismos de comunicação entre os modelos de execução de cenários e os modelos de representação dos ambientes.
- Estender a noção de recurso discreto ao caso dos sistemas de gerenciamento de *WorkFlow*, fazendo a distinção entre casos de um processo, recursos produzidos e consumidos e atores envolvidos na execução de atividades.
- Estender o uso de modelos baseados em rede de Petri para especificar e avaliar arquiteturas distribuídas de jogadores inteligentes usados em jogos de tabuleiros.

Os resultados do presente trabalho foram apresentados na IEEE International Conference on Systems, Man and Cybernetics por meio do artigo *Game modeling using WorkFlow nets* [de Oliveira et al. 2011].

Referências Bibliográficas

[Abrial 1996] Abrial, J.-R. (1996). *The B-book: assigning programs to meanings*. Cambridge University Press, New York, NY, USA.

[Allen e Garlan 1996] Allen, R. e Garlan, D. (1996). The Wright Architectural Specification Language. Technical report.

[Allen e Garlan 1992] Allen, R. J. e Garlan, D. (1992). A Formal Approach to Software Architectures. In van Leeuwen, J. (editor), *IFIP Congress (1)*, volume A-12 de *IFIP Transactions*, pp. 134–141. North-Holland.

[Back e von Wright 1990] Back, R.-J. e von Wright, J. (1990). Refinement Calculus, Part I: Sequential Nondeterministic Programs. In *Stepwise Refinement of Distributed Systems, Models, Formalisms, Correctness, REX Workshop*, pp. 42–66, London, UK. Springer-Verlag.

[BETHKE 2003] BETHKE, E. (2003). *Game Development and Production*. Wordware Publishing, first edition.

[Bi e Zhao 2004] Bi, H. H. e Zhao, J. L. (2004). Applying Propositional Logic to Workflow Verification. *Inf. Technol. and Management*, 5:293–318.

[Borger e Stark 2003] Borger, E. e Stark, R. F. (2003). *Abstract State Machines: A Method for High-Level System Design and Analysis*. Springer-Verlag New York, Inc., Secaucus, NJ, USA.

[Bruegge e Dutoit 2009] Bruegge, B. e Dutoit, A. H. (2009). *Object-Oriented Software Engineering Using UML, Patterns, and Java*. Prentice Hall Press, Upper Saddle River, NJ, USA, third edition.

[Cardoso e Valette 1997] Cardoso, J. e Valette, R. (1997). *Redes de Petri*. Editora da UFSC.

[Cardoso et al. 1999] Cardoso, J., Valette, R., e Dubois, D. (1999). Possibilistic Petri nets. *IEEE Transactions on Systems, Man, and Cybernetics*, 29:573–582.

[Champagnat et al. 2000] Champagnat, R., Pradin-Chézalviel, B., e Valette, R. (2000). Petri nets and linear logic as an aid for scheduling batch processes. In *ADPM 2000*. Shaker.

[Couvreur e Poitrenaud 2009] Couvreur, J.-M. e Poitrenaud, D. (2009). Petri Net Unfoldings - Properties. In Diaz, M. (editor), *Petri Nets: Fundamental Models, Verification and Applications*, pp. 315–434. Wiley-ISTE.

[David e Alla 2004] David, R. e Alla, H. (2004). *Discrete, Continuous, and Hybrid Petri Nets*. Springer, first edition.

[DAVISON 2005] DAVISON, A. (2005). *Killer Game Programming In Java*. OREILLY & ASSOC, first edition.

[de Oliveira et al. 2011] de Oliveira, G. W., Julia, S., e Passos, L. M. S. (2011). Game modeling using WorkFlow nets. In *SMC*, pp. 838–843. IEEE.

[Dijkstra 1975] Dijkstra, E. W. (1975). Guarded commands, nondeterminacy and formal derivation of programs. *Commun. ACM*, 18(8):453–457.

[Florin e Natkin 1984] Florin, G. e Natkin, S. (1984). Définition formelle des Réseaux de Petri Stochstiques. Technical report, CNAM - Paris.

[Floyd 1967] Floyd, R. W. (1967). Assigning meanings to programs. *Mathematical aspects of computer science*, 19(19-32):1.

[Gal et al. 2002] Gal, V., Prado, C. L., Natkin, S., Vega, L., e Cnam, C. (2002). Writing for video games. In *In Proceedings Laval Virtual (IVRC)*.

[Genrich 1987] Genrich, H. J. (1987). Predicate/transition nets. In *Advances in Petri nets 1986, part I on Petri nets: central models and their properties*, pp. 207–247, Bad Honnef, London, UK. Springer-Verlag.

[George e Vaughn 2003] George, V. e Vaughn, R. (2003). Application of lightweight formal methods in requirement engineering. In *Crosstalk: The J. Defense Softw. Eng.*

[Ghose 2000] Ghose, A. (2000). Formal Methods for Requirements Engineering. In *ISMSE*, pp. 13–16. IEEE Computer Society.

[Girard 1987] Girard, J.-Y. (1987). Linear logic. *Theoretical Computer Science*, 50(1):1–102.

[Girault 1997] Girault, F. (1997). *A logic for Petri nets*, volume 31. Eddition Hermes.

[Gochet e Gribomont 1990] Gochet, P. e Gribomont, P. (1990). *Logique: méthodes pour l'informatique fondamentale*, volume Vol 1. Hermès.

[Gold 2004] Gold, J. (2004). *Object-Oriented Game Development*. Addison Wesley.

[Hoare 1969] Hoare, C. A. R. (1969). An Axiomatic Basis for Computer Programming. *Commun. ACM*, 12(10):576–580.

[Hoare 2002] Hoare, C. A. R. (2002). Proof of correctness of data representations. pp. 385–396.

[Jean-Yves Girard e Regnier 1995] Jean-Yves Girard, Y. L. e Regnier, L. (1995). Cambridge University Press.

[Jensen 1991] Jensen, K. (1991). Coloured petri nets: A high level language for system design and analysis. pp. 342–416.

[Jones 1990] Jones, C. B. (1990). *Systematic Software Development using VDM*. Prentice-Hall.

[Julia 1998] Julia, S. (1998). Da concepção ao controle em tempo real de sistemas flexíveis de manufatura usando as Redes de Petri. In *XII Congresso Brasileiro de Automática*.

[Julia e Soares 2003] Julia, S. e Soares, M. S. (2003). Verification of Real Time UML specifications through a specialized inference mechanism based on a Token Player Algorithm and the sequent calculus of Linear Logic. In *Proceedings of the 15th European Simulation Symposium and Exhibition*, pp. 65–70, Delft, The Netherlands.

[Kaiser 2007] Kaiser, D. M. (2007). Automatic feature extraction for autonomous general game playing agents. In *AAMAS '07: Proceedings of the 6th international joint conference on Autonomous agents and multiagent systems*, pp. 1–7, Honolulu, Hawaii, New York, NY, USA. ACM.

[LAMOTHE 2002] LAMOTHE, A. (2002). *Tricks Of The Windows Game Programming Gurus*. SAMS, second edition.

[Lewinski 1999] Lewinski, J. S. (1999). *Developer's Guide to Computer Game Design*. Wordware Publishing Inc., Plano, TX, USA.

[Li e Song 2005] Li, S. e Song, B. (2005). Normalized workflow net (NWF-net): its definition and properties. *Future Gener. Comput. Syst.*, 21:1004–1014.

[LLOPIS 2003] LLOPIS, N. (2003). *C++ For Game Programmers*. CHARLES RIVER MEDIA, first edition.

[Loki Software e Hall 2001] Loki Software, I. e Hall, J. (2001). *Programming linux games*. No Starch Press Series. No Starch Press.

[Menasche 1982] Menasche, M. (1982). *Analyse des réseaux de Petri temporisés et application aux systèmes distribués*. PhD thesis, Univ. Paul Sabatier, Toulouse, France.

[Merlin 1974] Merlin, P. M. (1974). *A study of the recoverability of computing systems*. PhD thesis, University of California, Irvine. AAI7511026.

[Meyer 1991] Meyer, B. (1991). *Advances in Object-Oriented Software Engineering*, chapter Design by Contract, pp. 1–50. Prentice-Hall.

[Moncelet et al. 1998] Moncelet, G., Christensen, S., Demmou, H., Paludetto, M., e Porras, J. (1998). Dependability Evaluation Of A Simple Mechatronic System Using Coloured Petri Nets. In *Proceedings of Workshop on Practical Use of Coloured Petri nets and Design/CPN*.

[Morgan 1988] Morgan, C. (1988). The Specification Statement. *ACM Trans. Program. Lang. Syst.*, 10(3):403–419.

[Morris 1987] Morris, J. M. (1987). A Theoretical Basis for Stepwise Refinement and the Programming Calculus. *Sci. Comput. Program.*, 9(3):287–306.

[Murata 1989] Murata, T. (1989). Petri nets: Properties, analysis and applications. In *Proceedings of the IEEE*, Volume : 77 , Issue:4, pp. 541–580.

[Natkin et al. 2004] Natkin, S., Vega, L., De, C., e Cnam, R. I. (2004). A new methodology for spatiotemporal Game Design. In *Proceedings of CGAIDE*, pp. 109–113.

[Onder 2002] Onder, B. (2002). *Game Design Perspectives*, chapter Writing the adventure game, pp. 28–43. Charles River Media Game Development Series. Charles River Media.

[Passos 2009] Passos, L. M. S. (2009). Formalização de workflow nets utilizando lógica linear: análise qualitativa e quantitativa. Master's thesis, Faculdade de Computação, Universidade Federal de Uberlândia.

[Penton 2003] Penton, R. (2003). *Data Structures for Game Programmers*. Muska & Lipman/Premier-Trade.

[PENTON 2005] PENTON, R. (2005). *Beginning C# Game Programming*. COURSE TECHNOLOGY, first edition.

[Peterson 1981] Peterson, J. L. (1981). *Petri Net Theory and the Modeling of Systems*. Prentice Hall PTR, Upper Saddle River, NJ, USA.

[Pradin-Chezalviel et al. 1999] Pradin-Chezalviel, B., Valette, R., e Kunzle, L. A. (1999). Scenario Durations Characterization of T-Timed Petri Nets Using Linear Logic. In *PNPM '99: Proceedings of the The 8th International Workshop on Petri Nets and Performance Models*, p. 208, Washington, DC, USA. IEEE Computer Society.

[Ramchandani 1974] Ramchandani, C. (1974). *Analysis of assynchronous concurrent systems by timed Petri net Models*. PhD thesis, Massachusetts Institute of Technology, Cambridge, MA, USA.

[Riviere et al. 2001] Riviere, N., Valette, R., Pradin-Chezalviel, B., e Ups, I. A. . (2001). Reachability and Temporal Conflicts in t-Time Petri Nets. In *PNPM '01: Proceedings of the 9th international Workshop on Petri Nets and Performance Models (PNPM'01)*, pp. 229–, Washington, DC, USA. IEEE Computer Society.

[Rollins e Morris 2004] Rollins, A. e Morris, D. (2004). *Game Architecture and Design*. New Riders Games.

[Rouse 2004] Rouse, R. (2004). *Game Design: Theory and Practice*. Jones & Bartlett Publishers, second edition.

[Rucker 2003] Rucker, R. (2003). *Software Engineering and Computer Games*. Addison Wesley.

[Sanders 1998] Sanders, M. (1998). Constraint programming with object-oriented Petri nets. In *Systems, Man, and Cybernetics, 1998. 1998 IEEE International Conference on*, volume 1, pp. 289 – 294 vol.1.

[Scarpelli e Gomide 1993] Scarpelli, H. e Gomide, F. (1993). Fuzzy Reasoning and Fuzzy Petri Nets in Manufacturing Systems Modeling. *Journal of Intelligent and Fuzzy Systems*, 1(3):225–241.

[SIANG e RAO 2004] SIANG, A. e RAO, G. (2004). Designing interactivity in computer games: A UML Approach. *Int. J. Intelligent Games and Simulation 3*.

[Sifakis 1977] Sifakis, J. (1977). Use of Petri Nets for Performance Evaluation. In *Proceedings of the Third International Symposium on Measuring, Modelling and Evaluating Computer Systems*, pp. 75–93, Amsterdam, The Netherlands, The Netherlands. North-Holland Publishing Co.

[Silbertin 1985] Silbertin, C. (1985). High-Level Petri nets with Data Structures. In *European Workflow on Application and Theory of Petri Nets*. Digital Systems Laboratory. Helsink, Finland.

[Soares 2004] Soares, M. S. (2004). Uma abordagem baseada num jogador de redes de Petri p-temporal e no cálculo de sequentes da Lógica Linear para a verificação de cenários de Sistemas Tempo Real especificados através de diagramas dinâmicos da UML. Master's thesis, Faculdade de Computação, Universidade Federal de Uberlândia.

[Tazza 1987] Tazza, M. (1987). *Quantitative analysis of a resource allocation problem: a net theory based proposal*, pp. 511–532. Springer-Verlag New York, Inc., New York, NY, USA.

[V. Gal et al. 2002] V. Gal, Natkin, S., Vega, L., e Prado, C. L. (2002). Processus et outils utilisés pour la conception et la réalisation des jeux vidéo. In *Rapport CEDRIC*.

[Valette 1979] Valette, R. (1979). Analysis of Petri nets by stepwise refinements. *Journal of Computer and System Sciences*, 18:35–46.

[Valette 1980] Valette, R. R. (c1980.). *Systèmes de commande en temps réel :.* SCM„ Paris :.

[van der Aalst 1996] van der Aalst, W. (1996). Structural Characterizations of Sound Workflow Nets. *Computing Science Reports/23*, (96).

[van der Aalst e van Hee 2004] van der Aalst, W. e van Hee, K. (2004). *Workflow Management: Models, Methods, and Systems*. The MIT Press.

[van der Aalst 1997] van der Aalst, W. M. P. (1997). Verification of Workflow Nets. In *ICATPN'97 : Proceedings of the 18th International Conference on Application and Theory of Petri Nets*, pp. 407–426, London, UK. Springer-Verlag.

[van der Aalst 1998] van der Aalst, W. M. P. (1998). The Application of Petri Nets to Workflow Management. *Journal of Petri Net Circuits, Systems, and Computers*, 8(1):21–66.

[van Lamsweerde 2003] van Lamsweerde, A. (2003). From System Goals to Software Architecture. In Bernardo, M. e Inverardi, P. (editores), *SFM*, volume 2804 de *Lecture Notes in Computer Science*, pp. 25–43. Springer.

[Wolf 2007] Wolf, M. J. P. (2007). *The Video Game Explosion A History from PONG to PlayStation and Beyond*. Greenwood Press, first edition.

[Woodcock et al. 2009] Woodcock, J., Larsen, P. G., Bicarregui, J., e Fitzgerald, J. (2009). Formal methods: Practice and experience. *ACM Comput. Surv.*, 41(4):1–36.