Uma abordagem baseada em Redes de Petri para Modelagem, Análise e Simulação de Cenários de Vídeo Games Singleplayer e Multiplayer

Franciny Medeiros Barreto

Universidade Federal de Uberlândia
Faculdade de Computação
Programa de Pós-Graduação em Ciência da Computação

Uberlândia
2020

Franciny Medeiros Barreto

Uma abordagem baseada em Redes de Petri para Modelagem, Análise e Simulação de Cenários de Vídeo Games Singleplayer e Multiplayer

Tese de doutorado apresentada ao Programa de Pós-graduação da Faculdade de Computação da Universidade Federal de Uberlândia como parte dos requisitos para a obtenção do título de Doutor em Ciência da Computação.

Área de concentração: Ciência da Computação

Orientador: Prof. Dr. Stéphane Julia

Uberlândia
2020

Ficha Catalográfica Online do Sistema de Bibliotecas da UFU
com dados informados pelo(a) próprio(a) autor(a).

B273 Barreto, Franciny Medeiros, 1992-
2020 Uma abordagem baseada em Redes de Petri para Modelagem, Análise e Simulação de Cenários de Vídeo Games Singleplayer e Multiplayer [recurso eletrônico] / Franciny Medeiros Barreto. - 2020.

Orientador: Stéphane Julia.
Tese (Doutorado) - Universidade Federal de Uberlândia, Pós-graduação em Ciência da Computação.
Modo de acesso: Internet.
Disponível em: http://doi.org/10.14393/ufu.te.2020.754
Inclui bibliografia.

1. Computação. I. Julia, Stéphane ,1969-, (Orient.). II. Universidade Federal de Uberlândia. Pós-graduação em Ciência da Computação. III. Título.

CDU: 681.3

Bibliotecários responsáveis pela estrutura de acordo com o AACR2:

Gizele Cristine Nunes do Couto - CRB6/2091

UNIVERSIDADE FEDERAL DE UBERLÂNDIA
Coordenação do Programa de Pós-Graduação em Ciência da Computação
Av. João Naves de Ávila, nº 2121, Bloco 1A, Sala 243 - Bairro Santa Mônica, Uberlândia-MG, CEP 38400-902
Telefone: (34) 3239-4470 - www.ppgco.facom.ufu.br - cpgfacom@ufu.br

ATA DE DEFESA - PÓS-GRADUAÇÃO

Programa de Pós-Graduação em:	Ciência da Computação				
Defesa de:	Tese de doutorado, 31/2020, PPGCO				
Data:	19 de novembro de 2020	Hora de início:	14:03	Hora de encerramento:	17:18
Matrícula do Discente:	11523CCP002				
Nome do Discente:	Franciny Medeiros Barreto				
Título do Trabalho:	Uma abordagem baseada em Redes de Petri para Modelagem, Análise e Simulação de Cenários de Vídeo Games Singleplayer e Multiplayer				
Área de concentração:	Ciência da Computação				
Linha de pesquisa:	Engenharia de Software				
Projeto de Pesquisa de vinculação:	-				

Reuniu-se, por videoconferência, a Banca Examinadora, designada pelo Colegiado do Programa de Pós-graduação em Ciência da Computação, assim composta: Professores Doutores: Fernanda Francielle de Oliveira Malaquias - FAGEN/UFU; Márcia Aparecida Fernandes - FACOM/UFU; José Reinaldo Silva - USP; Ricardo Luders - UTFPR e Stéphane Julia - FACOM/UFU, orientador da candidata.

Os examinadores participaram desde as seguintes localidades: José Reinaldo Silva - São Paulo/SP; Ricardo Luders - Curitiba/PR; Fernanda Francielle de Oliveira Malaquias, Márcia Aparecida Fernandes e Stéphane Julia - Uberlândia/MG. A discente participou da cidade de Jataí/GO.

Iniciando os trabalhos o presidente da mesa, Prof. Dr. Stéphane Julia, apresentou a Comissão Examinadora e a candidata, agradeceu a presença do público, e concedeu à Discente a palavra para a exposição do seu trabalho. A duração da apresentação da Discente e o tempo de arguição e resposta foram conforme as normas do Programa.

A seguir o senhor presidente concedeu a palavra, pela ordem sucessivamente, aos examinadores, que passaram a arguir a candidata. Ultimada a arguição, que se desenvolveu dentro dos termos regimentais, a Banca, em sessão secreta, atribuiu o resultado final, considerando a candidata:

Aprovada.

Esta defesa faz parte dos requisitos necessários à obtenção do título de Doutor.

O competente diploma será expedido após cumprimento dos demais requisitos, conforme as normas do Programa, a legislação pertinente e a regulamentação interna da UFU.

Nada mais havendo a tratar foram encerrados os trabalhos. Foi lavrada a presente ata que após lida e achada conforme foi assinada pela Banca Examinadora.

Documento assinado eletronicamente por **Ricardo Luders, Usuário Externo**, em 20/11/2020, às 15:54, conforme horário oficial de Brasília, com fundamento no art. 6º, § 1º, do Decreto nº 8.539, de 8 de outubro de 2015.

Documento assinado eletronicamente por **Márcia Aparecida Fernandes, Professor(a) do Magistério Superior**, em 20/11/2020, às 17:53, conforme horário oficial de Brasília, com fundamento no art. 6º, § 1º, do Decreto nº 8.539, de 8 de outubro de 2015.

Documento assinado eletronicamente por **Stéphane Julia, Professor(a) do Magistério Superior**, em 21/11/2020, às 12:13, conforme horário oficial de Brasília, com fundamento no art. 6º, § 1º, do Decreto nº 8.539, de 8 de outubro de 2015.

Documento assinado eletronicamente por **Jose Reinaldo Silva, Usuário Externo**, em 22/11/2020, às 22:31, conforme horário oficial de Brasília, com fundamento no art. 6º, § 1º, do Decreto nº 8.539, de 8 de outubro de 2015.

Documento assinado eletronicamente por **Fernanda Francielle de Oliveira Malaquias, Professor(a) do Magistério Superior**, em 23/11/2020, às 10:08, conforme horário oficial de Brasília, com fundamento no art. 6º, § 1º, do Decreto nº 8.539, de 8 de outubro de 2015.

A autenticidade deste documento pode ser conferida no site https://www.sei.ufu.br/sei/controlador_externo.php?acao=documento_conferir&id_orgao_acesso_externo=0, informando o código verificador **2397610** e o código CRC **1D95DDB7**.

Referência: Processo nº 23117.068955/2020-12 SEI nº 2397610

Dedico este trabalho a minha mãe e ao meu irmão, Hugo.

Agradecimentos

Agradeço a Deus, por tudo que tem feito por mim e por me permitir caminhar ao lado de pessoas tão queridas.

Agradeço a minha família pelo apoio em todos os momentos. A minha mãe, Fátima, agradeço pela dedicação e amor incondicional, e aos meus irmãos, Amaro, Victor e Hugo, agradeço pelo carinho e amizade. As minhas sobrinhas, Júlia e Sophia, agradeço por sempre iluminar meus dias com momentos de muita felicidade.

Agradeço as amigas Fabíola, Fernanda e Joslaine por tornarem o meu caminho muito mais agradável. Aos amigos Túlio e Gabriel agradeço pelo apoio e carinho. Agradeço as minhas amigas Myllene e Sara pela amizade, sinceridade e parceria desde sempre.

Agradeço ao meu orientador, professor Dr. Stéphane Julia, por todos esses anos de orientação na pós-graduação. Agradeço por todos os seus ensinamentos que me ajudaram a amadurecer como profissional, pela paciência e compreensão em muitos momentos que foram difíceis para mim e, principalmente, por sempre me motivar a chegar até o final.

Por fim, a todos os professores do programa de pós-graduação em Ciência da Computação da UFU agradeço pelos ensinamentos que contribuíram para a minha formação. Agradeço aos secretários da pós-graduação, Erisvaldo e Sônia, pelo profissionalismo impecável e pelo cuidado em sempre me ajudar quando precisei.

"O que quer que você faça na sua vida será insignificante, mas é muito importante que você faça, porque ninguém mais o fará."

(Mahatma Gandhi)

Resumo

Este trabalho tem como objetivo apresentar uma abordagem para a modelagem e análise formal de cenários de vídeo games, usando para isso o formalismo das redes de Petri. Nesta pesquisa são considerados jogos com um único jogador (*singleplayer*) e jogos com vários jogadores (*multiplayer*). Um cenário de vídeo game é considerado como sendo uma sequência de ações realizadas pelo jogador. A representação das ações que o jogador precisa executar, a fim de alcançar o objetivo do jogo, pode ser representada por meio de uma WorkFlow net. O mundo virtual onde o jogo acontece pode ser descrito através de um mapa topológico representado por um tipo particular de rede de Petri denominado grafo de estados. Ambos modelos são formalmente definidos e denominados, respectivamente, de Modelo Lógico e de Modelo Topológico. Os mecanismos de comunicação que existem entre os dois modelos (Lógico e Topológico) são especificados por meio de redes de Petri Coloridas e implementados na ferramenta CPN Tools. A versão temporizada do Modelo Lógico apresenta as durações mínimas, médias e máximas das atividades de um cenário de jogo por meio de distribuições de tempo aleatórias exponenciais. A versão temporizada do Modelo Topológico apresenta os tempos mínimos e máximos que um jogador leva para acessar as diversas áreas do mundo virtual por meio de distribuições de tempo aleatórias uniformes. Baseado nos modelos Lógicos e Topológicos temporizados, um método de simulação é proposto para estimar as durações médias dos tempos de jogos. A segunda parte proposta neste trabalho diz respeito à modelagem formal de cenários de jogos no caso multiplayer. Para isso foi definido um novo Modelo Lógico baseado em Workflow nets Possibilísticas que permitem representar as possibilidades de interação entre jogadores e as consequências de tais interações em termos de jogabilidade. O Modelo Topológico no caso multiplayer é baseado nos grafos de estados e permite que as áreas do mundo virtual sejam compartilhadas entre dois ou mais jogadores. Uma versão implementada dos modelos é apresentada na forma de uma rede de Petri Colorida interpretada na ferramenta CPN Tools. Também é proposto um método para o estudo da jogabilidade que utiliza o monitoramento das atividades e a simulação por replicação para diferentes cenários. Dessa forma, é possível estimar o tempo de jogo multiplayer bem como o papel

de cada jogador no jogo. Os jogos Silent Hill II e Tom Clancy's Ghost Recon: Wildlands servem de exemplos ilustrativos para validar as abordagens propostas neste trabalho.

Palavras-chave: Redes de Petri Coloridas. Redes de Petri Possibilísticas. Redes de Petri Temporizadas. Workflow nets. Grafos de estados. CPN Tools..

Abstract

This work aims to present an approach for the modeling and formal analysis of videogame scenarios, using the formalism of Petri nets. This work considers singleplayer games and multiplayer games. A videogame scenario is considered to be a sequence of actions performed by the player. The representation of the actions that the player needs to perform in order to achieve the objective of the game can be represented by means of a WorkFlow net. The virtual world where the game takes place can be described through a topological map represented by a particular type of Petri net called state graph. Both models are formally defined and named, respectively, Logical Model and Topological Model. The communication mechanisms that exist between the two models (Logical and Topological) are specified through Colored Petri nets and implemented using CPN Tools. The timed version of the Logical Model presents the minimum, average and maximum activity durations of a game scenario through exponential random time distributions. The timed version of the Topological Model shows the minimum and maximum times it takes a player to access the various areas of the virtual world through uniform random time distributions. Based on the timed Logical and Topological models, a simulation method is proposed to estimate the average durations of the game times. The second part proposed in this work concerns the formal modeling of game scenarios in the multiplayer case. For this, a new Logical Model based on Possibilistic Workflow nets is defined that allows to represent the possibilities of interaction between players and the consequences of such interactions in terms of gameplay. The Topological Model in the multiplayer case is based on the state graphs and allows the areas of the virtual world to be shared between two or more players. An implemented version of the model is produced in the form of a Colored Petri net interpreted using CPN Tools. A method for the study of gameplay is also proposed using activity monitors and simulation replication considering different scenarios. Thus, it is possible to estimate the duration of a multiplayer game as well as the role of each player in the game. The games Silent Hill II and Tom Clancy's Ghost Recon: Wildlands serve as illustrative examples to validate the approaches proposed in this work.

Keywords: Colored Petri nets. Possibilistic Petri nets. Timed Petri nets. Workflow nets. State Graphs. CPN Tools..

Lista de ilustrações

Lista de tabelas

Sumário

Capítulo 1

Introdução

A discussão sobre a importância dos jogos não é algo novo na literatura. Os jogos possuem uma longa história no desenvolvimento de quase todas as culturas e sociedades (HUIZINGA, 2000). Com o advento de novas tecnologias, o debate sobre o uso de atividades de jogos estendeu-se aos jogos eletrônicos. Crawford (1984) afirma que um jogo é um sistema fechado e formal que representa subjetivamente um subconjunto da realidade. Por fechado, entende-se que o jogo é completo e autossuficiente em sua estrutura. Por formal, significa que o jogo tem regras explícitas. O termo sistema é usado pelo autor devido ao fato de que um jogo é uma coleção de partes que interagem umas com as outras, muitas vezes de maneiras complexas. Para Kanode Christopher M. e Haddad (2009), um jogo é uma síntese de código, imagens, música e atuação que se reúnem em forma de entretenimento. De modo geral, pode-se dizer que os jogos são um tipo especial de aplicação multimídia que requer a participação ativa do usuário.

Para Kanode Christopher M. e Haddad (2009), no entanto, as aplicações de jogos diferem de *softwares* tradicionais não apenas pelo uso e integração de recursos multimídias, como também pelas suas fases de pré-produção. A fase de pré-produção é um tipo de coleta de requisitos e criação de protótipos do jogo. Do ponto de vista clássico da indústria de vídeo games, o desenvolvimento de jogos é composto por duas etapas principais denominadas de *game design* e *level design* (GAL et al., 2002). Nessas duas etapas são elaborados e descritos aspectos importantes e decisivos para a história do jogo.

Na etapa de *game design* são definidos os aspectos principais do universo do jogo e cada detalhe de como ele irá funcionar (GAL et al., 2002). Nessa fase, define-se o contexto do jogo (época, estilo, referências histórias, etc.), o cenário global (topologia, gráficos de navegação, personagens principais, etc.), as características principais que tornam o jogo único, os princípios de jogabilidade (objetivo a ser alcançado, por exemplo), os princípios de ergonomia, e as imagens e sons do jogo. Em outras palavras, a natureza do jogo é determinada na fase de *game design*.

De acordo com Gal et al. (2002), é na etapa de *level design* que todos os diferentes componentes do jogo se unem. Nessa fase, define-se a descrição de como o jogador irá

interagir com os objetos (itens) do jogo, a descrição das missões que o jogador executará, bem como o nível de dificuldade para a execução dessas tarefas. No *game* e *level design* são elaborados e descritos aspectos importantes e decisivos para a história do jogo. Portanto, essas duas etapas do processo de criação de jogos são fundamentais para o sucesso do projeto.

O processo de criação de vídeo games depende diretamente das habilidades criativas do designer, das influências que ele tem (por exemplo, livros, filmes, música, cultura, etc.) e também de jogos existentes que funcionam como uma base para novas ideias (ALMEIDA; SILVA, 2013). De acordo com Collé, Champagnat e Prigent (2005), os designers de vídeo games tendem a criar mais jogos baseados em cenários, com o objetivo de fazê-los mais atrativos ao público. Em termos de *game design*, um cenário corresponde à descrição clássica de uma sequência de ações das principais fases do jogo, e de como ocorre a navegação entre essas fases. Já no *level design*, os cenários correspondem ao posicionamento dos objetos em relação às missões do jogo.

Neste contexto, Rollings e Morris (2003) afirmam que é preciso criar uma (ou mais) série de desafios casualmente ligados em um ambiente simulado. É importante propor desafios que não sejam nem fáceis e nem difíceis demais para se resolver. É de fundamental importância garantir que a experiência do jogo leve a uma sucessão de metas dentro de um prazo razoável. Além disso, o jogador precisa ter a sensação de liberdade no mundo virtual, mesmo quando ele é orientado para uma solução de uma forma inconsciente (ROLLINGS; MORRIS, 2003). A criação desses desafios não é uma tarefa meramente trivial (NATKIN; VEGA; GRÜNVOGEL, 2004). Segundo Almeida e Silva (2013), os jogos são muito imprevisíveis para serem imaginados pelo designer até que sejam criados. De acordo com os autores, os jogos são o que os cientistas conhecem como sistemas complexos.

De acordo com Neil (2012), os designers de jogos, ao contrário da maioria dos outros profissionais de design, normalmente não usam ferramentas para projetar. A ludologia (o estudo dos jogos em geral, particularmente dos vídeo games) aponta a necessidade de criar modelos para especificar os mecanismos dos jogos, pois a falta deste conhecimento tem sido um dos grandes problemas do tradicional projeto de jogos (REYNO; CUBEL, 2009). Kanode Christopher M. e Haddad (2009) afirmam que um dos maiores problemas do desenvolvimento de jogos é a utilização de metodologias pobres, fazendo com que projetos não sejam finalizados no tempo estimado e com custos elevados. Além disso, Neil (2012) argumenta que as dinâmicas algorítmicas e interativas em tempo real do jogo são muito complexas para serem modeladas ou avaliadas com sucesso no papel, ou mesmo na mente de um designer.

Portanto, ferramentas formais e abstratas para projetar a jogabilidade (sejam modelos conceituais ou software) devem ser desenvolvidas para o projeto de vídeo game atingir os níveis desejados de sofisticação e expressão criativa (NEIL, 2012). É necessário encon-

trar novos conceitos e ferramentas para suprir os desafios do desenvolvimento de jogos. Neste sentido, a Engenharia de Software se torna indispensável para ajudar a suprir os desafios de desenvolvimento de jogos, melhorando a gerência dos projetos, diminuindo os riscos e garantindo o sucesso futuro da indústria de jogos (KANODE CHRISTOPHER M. E HADDAD, 2009).

1.1 Motivação

Em softwares e sistemas de engenharia é comum construir diagramas e gráficos para especificar visualmente conjuntos de requisitos. No contexto do processo de criação de vídeo games, os modelos visuais são mais sintéticos, naturalmente comunicativos e têm melhor escala. A prática da diagramação força os designers a extrair a essência da jogabilidade em poucos elementos visuais (ALMEIDA; SILVA, 2013).

Alguns trabalhos apresentam o uso de diagramas para representar o conjunto de requisitos de um jogo. Os diagramas UML (*Unified Modelling Language* - Linguagem de Modelagem Unificada), por exemplo, têm sido usados para mostrar como os diferentes objetos em um jogo irão interagir de acordo com algumas ações que serão executadas pelo jogador ((ANG; RAO, 2004) e (RUCKER, 2003)). De acordo com Oliveira, Julia e Passos (2011), diagramas UML são interessantes a medida que produzem uma estrutura de execução do jogo. No entanto, eles não apresentam de maneira explícita os possíveis cenários que existem em uma missão ou nível de jogo.

Outra abordagem utilizada para a modelagem de jogos é a de métodos formais. Os métodos formais são técnicas matemáticas, frequentemente suportadas por ferramentas para desenvolvimento de sistemas de software e hardware. Os modelos formais são rigorosamente definidos e não contém especificações ambíguas (LEWIS; WHITEHEAD, 2011). Por meio de modelos formais, é possível que o projetista explore o projeto do jogo usando técnicas manuais ou automáticas para simular as passagens entre os níveis do jogo e verificar certas propriedades esperadas. Dentro deste contexto, alguns trabalhos já consideram as redes de Petri como uma ferramenta eficiente para modelagem e análise de sistemas de jogos como os trabalhos apresentados em (FINKBEINER; OLDEROG, 2017) e (PEÑA et al., 2016).

As redes de Petri são consideradas como uma ferramenta gráfica e matemática de representação formal que permite modelagem, análise e controle de sistemas a eventos discretos que comportam atividades paralelas, concorrentes e assíncronas (MURATA, 1989). No trabalho de Araújo e Roque (2009), por exemplo, diagramas baseados em redes de Petri foram utilizados para representar as possíveis ações dos jogadores em relação aos objetivos do jogo. Assim, o fluxo do jogo pôde ser mapeado e simulado pelas redes de Petri. Araújo e Roque (2009) afirmam que a simulação do comportamento de jogo oferece a possibilidade de detectar problemas ainda na fase de projeto.

No trabalho de Natkin, Vega e Grünvogel (2004), a modelagem de cenários de jogos foi baseada em um novo tipo de rede de Petri denominado rede de transações. Uma rede de transações permite modelar transações lógicas e temporais enquanto um mapa topológico do jogo é modelado por um tipo de grafo, chamado hiper-grafo. Neste tipo de grafo, as arestas são criadas dinamicamente. Mecanismos de comunicação entre os dois modelos tentam estabelecer a influência que um modelo tem sobre o outro. Porém, a análise formal é aplicada apenas na rede de transações (representada por uma rede de Petri). Assim, a validação de cenários de jogos não pode ser formalmente realizada usando ferramentas de análise de redes de Petri, devido à semântica operacional das redes de transação e dos hiper-grafos não serem as mesmas.

Já em Oliveira, Julia e Passos (2011), é apresentada uma abordagem baseada em um tipo particular de rede de Petri, denominado WorkFlow net (AALST; HEE, 2004), para especificar cenários existentes em um jogo. Nesta abordagem, uma WorkFlow net representa as atividades que devem ser executadas pelo jogador a fim de alcançar um objetivo específico no jogo. O conjunto dessas atividades formam então um cenário de jogo. Para validar os possíveis cenários que o jogador pode executar em uma missão de um jogo é utilizado o cálculo do sequente da lógica linear. O cálculo do sequente prova a corretude de um critério específico que valida tais cenários (OLIVEIRA; JULIA; PASSOS, 2011).

O trabalho de Lee e Cho (2012) aborda a criação de um mecanismo para gerar enredo de *quest* consistindo na representação de eventos baseados em redes de Petri. Cada elemento da rede representa uma característica do jogo. Por exemplo, os lugares indicam uma pré-condição para uma ação, as transições representam as ações do jogador, e os arcos representam relacionamentos casuais entre ação e pré-condição. Segundo Lee e Cho (2012), as redes de Petri ajudam a identificar elementos passivos (tais como as condições) e elementos ativos (tais como as ações) da história, facilitam a representação de eventos independentes, assim como a representação de restrições e a sincronização de eventos.

Em Barreto e Julia (2014), as redes de Petri foram utilizadas para modelar as atividades existentes em um nível de jogo, bem como o mapa topológico correspondente do mundo virtual. Usando o mesmo formalismo, os autores adicionaram um mecanismo de comunicação entre os dois modelos (modelo de atividades e modelo de mapa topológico), obtendo assim um modelo global baseado numa rede de Petri única. Tal modelo foi analisado considerando-se certas propriedades que garantem a consistência da correta execução das atividades e da correta disposição das diversas áreas existentes no mundo virtual de um jogo.

O conceito de redes de Petri na modelagem de jogos também foi apresentado no trabalho de Reuter, Göbel e Steinmetz (2015) para apresentar uma abordagem que atua nos estágios iniciais do projeto de jogo, detectando erros estruturais em jogos *singleplayer* (um jogador) e multiplayer (vários jogadores). Os autores utilizaram as redes de Petri

Coloridas para atribuir diferentes tipos as fichas e funções no modelo. Os elementos do jogo são mapeados para construtores apropriados das redes de Petri. A abordagem de Reuter, Göbel e Steinmetz (2015) utiliza o software CPN Tools para identificar erros estruturais como *deadlocks* (situações nas quais os jogadores não podem mudar o estado do jogo), *livelocks* (situações nas quais os jogadores podem mudar o estado do jogo, mas não podem alcançar o final), cenas inalcançáveis (situações nas quais os jogadores não podem chegar a uma cena sob quaisquer circunstâncias) e ações impossíveis (situações nas quais as ações nunca podem ser acionadas, devido a condições não satisfeitas).

Em Rezin et al. (2018), os autores apresentaram uma abordagem para verificar formalmente jogos de computador do tipo *multiplayer*. O processo do jogo é representado por um autômato finito não determinístico que evolui a cada unidade de tempo com base nas ações dos jogadores ou em ações aleatórias integradas ao jogo. Devido à complexidade da construção de um autômato, Rezin et al. (2018) desenvolveram uma ferramenta que possa apoiar a construção e representação formal do modelo de jogo e depois verificar automaticamente as propriedades de interesse. Após a verificação, o modelo pode ser automaticamente traduzido para uma linguagem de programação de alto nível a fim de integrá-lo ao jogo em desenvolvimento.

Os trabalhos citados acima limitaram-se à representação do fluxo de atividades do jogo e da topologia do mundo virtual correspondente. Além disso, a maioria considera apenas a modelagem de jogos com um único jogador. Araújo e Roque (2009) afirmam que em jogos com mais de um jogador, a complexidade imposta pelos eventos concorrentes do jogo pode representar modelos de cenários ainda mais complexos, e as propriedades dinâmicas são tipicamente mais difíceis de prever. Os trabalhos que consideram a representação de jogo *multiplayer* criaram modelos complexos que precisam passar por processos de otimização para serem analisados formalmente. Assim, a modelagem baseada em redes de Petri pode ser especialmente importante para simular jogos com vários jogadores sem inserir tanta complexidade no modelo como, por exemplo, o problema da explosão de estados que as abordagens de Reuter, Göbel e Steinmetz (2015) e Rezin et al. (2018) apresentam.

1.2 Objetivos e Desafios da Pesquisa

Os jogos, em geral, podem ser divididos em uma parte lógica e uma parte topológica. A parte lógica do jogo descreve a presença de cenários por meio da sequência lógica das atividades que precisam ser executadas no jogo a fim de se alcançar algum objetivo específico. A segunda parte corresponde à topologia no mundo virtual, onde as atividades propostas pelo jogo são executadas. Assim, é preciso definir um modelo para representar a estrutura lógica do jogo. É importante considerar neste modelo a representação de todas as atividades do jogo e, principalmente, o fluxo para execução dessas atividades. Também é preciso definir um modelo que seja capaz de representar a topologia do mundo

virtual. Tal estrutura é necessária para representar a evolução do jogador de acordo com a execução das atividades e com a interação entre o jogador e os elementos do mundo virtual. Criando tais modelos separadamente é possível que cada um seja verificado e analisado de forma independente. Por se tratar de modelos distintos, ainda é possível que alterações sejam realizadas em um modelo sem que também seja necessário alterar o outro modelo existente. Uma vez que ambos os modelos estão definidos, é preciso considerar que a junção dos dois é importante para simular o comportamento do jogo sob uma perspectiva global. Para tanto, é necessário definir um mecanismo a fim de estabelecer a comunicação entre os dois modelos. Um método de análise também precisa ser estabelecido para validar os modelos propostos.

A narrativa é um dos pontos centrais de um jogo. É por meio dela que os objetivos de jogo são definidos e as atividades estabelecidas. Algumas dessas atividades podem exigir que o jogador encontre objetos dispersos no jogo, colete pistas e resolva enigmas, ou ainda que interaja com personagens do próprio jogo. Alguns jogos permitem ainda que atividades deste tipo sejam executadas mais de uma vez, caso o jogador não consiga concluir um nível de jogo na primeira tentativa. Posto isto, é preciso então definir um modelo que permite a representação deste tipo de situação, onde uma mesma atividade pode ser realizada várias vezes de acordo com alguma condição pré-estabelecida. É importante levar em consideração que as atividades do jogo possuem um tempo médio de execução. A duração de uma atividade pode variar de acordo com a complexidade da ação que precisa ser executada e pode também depender do nível de habilidade do jogador. É preciso então definir um modelo que seja capaz de atribuir a cada atividade do jogo uma duração média. Além disso, também é necessário considerar um modelo que representa o tempo que um jogador leva para se locomover entre as diferentes áreas do mundo virtual. Dessa forma, o tempo total do jogo depende da soma dos tempos de cada atividade e do tempo gasto para percorrer as áreas do mapa. Ao final da execução de todas as atividades, o modelo precisa ser capaz de representar qual foi a duração média do jogo. Assim, a definição de um método para simulação dos cenários de jogos com a presença de tempo de jogo se faz necessária.

Quando se considera modelos em que há apenas um jogador ativo, o fluxo de execução das atividades não sofre alterações e o jogador poderá segui-lo de maneira esperada do início ao fim do jogo. Porém, quando mais de um jogador está ativo ao mesmo tempo no jogo, a execução das atividades podem sofrer alterações devido as diversas interações que os jogadores fazem entre si e no ambiente do jogo. Por exemplo, se um item único (como uma chave) é recuperado por um jogador, isso implica que os outros jogadores não poderão recuperar o mesmo item e por consequência não poderão realizar uma atividade específica do jogo (como abrir uma porta, por exemplo). Além disso, áreas do mundo virtual podem ser liberadas ou bloqueadas por um jogador com o intuito de prejudicar outros jogadores no mesmo ambiente. A tomada de decisão dos jogadores faz com que o

cenário do jogo fique cada vez mais complexo. O modelo para representar esse novo tipo de cenário precisa ser consistente com a execução real do jogo. Sendo assim, é necessário definir um modelo específico para a representação de cenários em jogos *multiplayer* que consiga gerenciar as interações entre os diversos jogadores existentes ao mesmo tempo no jogo.

Dentro deste contexto, a pesquisa descrita neste documento de qualificação tem como objetivo apresentar métodos para a análise e verificação formal de cenários de jogos *single-player* e *multiplayer* utilizando para isso o formalismo das redes de Petri. Especificamente, objetiva-se:

1. definir um modelo para a representação, análise e verificação formal de cenários de vídeo games, utilizando para isso o formalismo das redes de Petri. Para tanto, é preciso estabelecer um modelo lógico (para representar as atividades), um modelo topológico (para representar a topologia do mundo virtual) e um modelo de comunicação (para estabelecer as interações existentes entre os modelos);

2. estabelecer um modelo temporizado de cenários de vídeo games a partir do qual será possível estimar a duração média de um jogo. É preciso então definir funções de temporização com o objetivo de gerar tempos aleatórios associados aos modelos lógico e topológico. Com isso será então possível estabelecer diversos tipos de durações de execução de jogo (mínima/média/máxima) ainda em fase de projeto, ou seja, antes do jogo ser implementado;

3. implementar um modelo usando redes de Petri que seja capaz de representar cenários de vídeo games *multiplayer*. O modelo estabelecido irá considerar as interações/colaborações entre diversos jogadores que querem alcançar um objetivo em comum. Assim, será possível representar a influência que um jogador tem sobre o outro quando são consideradas atividades de jogo compartilhadas por mais de um jogador;

4. definir um método de análise qualitativa para avaliar a jogabilidade do vídeo game, bem como um método de análise quantitativa para estabelecer a estimação de tempo de jogo no caso *multiplayer*. Para ambos tipos de análise serão consideradas propriedades específicas relacionadas à teoria das redes de Petri.

1.3 Hipóteses

Diante dos objetivos descritos na seção anterior, as questões de pesquisa (QP) que norteiam este trabalho são:

QP 1 - É possível, usando um único formalismo, estabelecer modelos capazes de descrever o sequenciamento de atividades em um jogo em conjunto com a evolução do mundo virtual?
Hipótese 1 - Usando modelos formais baseados na teoria das redes de Petri, especificamente os tipos especiais de redes de Petri como as WorkFlow nets e os grafos de estados, é possível descrever as características principais do fluxo de atividades existente em um vídeo game, bem como a topologia do mundo virtual.
QP 2 - Será que a noção de jogabilidade de um vídeo game poderá ser verificada por meio dos modelos formais baseados em redes de Petri?
Hipótese 2 - É possível especificar um método de verificação e validação formal baseado nas boas propriedades das redes de Petri que pode ser aplicado à cenários de vídeo games, de tal modo a garantir o bom funcionamento do jogo em termos de sua jogabilidade.

Um fator que contribui para a análise de jogabilidade em um vídeo game é a duração média do jogo. Isso se deve ao fato de que é necessário criar desafios nos jogos que sejam interessantes e prendam a atenção do jogador, ao mesmo tempo que não sejam complexos demais a ponto de se tornar um fator para impedir o divertimento do jogador. Com isso, a seguinte questão de pesquisa é levantada.
QP 3 - Como representar a predição dos tempos mínimo, médio e máximo que um jogador levaria para completar uma atividade e, consequentemente, todas as atividades de um determinado vídeo game?
Hipótese 3 - As atividades de um vídeo game, incluindo a capacidade do jogador de percorrer vários lugares do mundo virtual, possuem, individualmente, um valor de duração associado. Tal valor pode ser representando em uma rede de Petri utilizando o conceito de temporização, em particular associando funções de tempo aleatório aos elementos ativos do modelo. Tais funções podem ter como parâmetro o nível de experiência/habilidade do jogador e assim gerar durações de atividades consistentes com a realidade de cada jogador.

A essência dos jogos, desde o início de sua história na sociedade, tem natureza coletiva (ZAGAL; NUSSBAUM; ROSAS, 2000). Com a evolução das tecnologias ao longo dos anos é razoável de se pensar que os vídeo games têm se tornado sistemas mais complexos a fim de se aproximar cada vez mais da realidade. Nesse sentido, os jogos *multiplayer* se apresentam como sistemas desafiadores a serem construídos, pois incluem em sua narrativa atividades compartilhadas, interações entre seus usuários e eventos que podem acontecer devido as decisões tomadas por cada jogador. Pode-se considerar então que em um jogo *multiplayer*, as atividades e os objetos do jogo são compartilhados entre vários jogadores. Portanto, as ações de um jogador irão interferir nas ações de outros jogadores em um determinado cenário de jogo. Sendo assim, enuncia-se a seguinte questão de pesquisa:
QP 4 - É possível estabelecer modelos formais concorrentes de redes de Petri que mostrarão eficientemente as colaborações e interações de um grupo finito de jogadores que

compartilham um objetivo de jogo em comum?

Hipótese 4 - Utilizando modelos baseado em redes de Petri é possível representar de maneira efetiva o comportamento dos cenários de vídeo games *multiplayer* por meio do conceito de incerteza inserido em uma versão Possibilística das redes de Petri, explorando então os conceitos de marcações imprecisas e disparos incertos para o modelo de atividades e de representação topológica.

A partir da quarta hipótese, pergunta-se:

QP 5 - Baseado em tal modelo, será então possível estimar aspectos relacionados à jogabilidade efetiva do vídeo game, em particular em relação à corretude da arquitetura do jogo e ao tempo de jogo global quando se considera um grupo de jogador em vez de um jogador único?

Hipótese 5 - Considerando os modelos construídos sob a perspectiva das rede de Petri Possibilísticas, bem como a inserção de temporização em tais modelos, é factível a apresentação de um método de verificação e validação formal que pode ser aplicado a cenários de vídeo games do tipo *multiplayer*, com o objetivo de garantir o bom funcionamento do jogo em termos de sua jogabilidade. Para tanto, é possível considerar as propriedades estruturais das redes de Petri e, além disso, se basear em ferramentas que permitem a implementação, análise e simulação de modelos baseados na teoria das redes de Petri.

1.4 Contribuições

A realização desta pesquisa contribui com a área de Engenharia de Software no que diz respeito ao processo de desenvolvimento de vídeo games, principalmente em suas etapas iniciais denominadas *game* e *level design*, onde são elicitados os requisitos e definida a arquitetura do sistema. A principal contribuição deste trabalho é a de propor métodos para a modelagem e análise formal de vídeo games usando para isso um único formalismo, especificamente o formalismo das redes de Petri. Tal método dará suporte a representação de aspectos importantes quando se considera a etapa de engenharia de requisitos de um jogo, em termos de sua jogabilidade. Além disso, um método para a análise formal dos modelos contribui para a identificação de problemas e falhas nos jogos antes que eles de fato sejam implementados.

A partir dos objetivos de pesquisa e das hipóteses apresentadas neste trabalho, são contribuições desta pesquisa:

- um método para a formalização da representação de cenários de vídeo games *singleplayer* considerando o sequenciamento lógico das ações do jogador e aspectos da topologia do mundo virtual;
- um método para a definição de modelos de cenários de vídeo games capaz de estimar a duração mínima, média e máxima de um jogo, considerando roteiros alternativos

que apresentam iterações necessárias quando o jogador falhar ao executar atividades específicas do jogo, bem como a capacidade que o jogador possui de ir e vir entre as diversas áreas do mundo virtual;

- ❏ um método para a formalização de cenários de vídeo games *multiplayer* capaz de representar as diversas interações entre os jogadores, e de lidar com a ocorrência de eventos inesperados no modelo quando se considera as decisões individuais de cada jogador que podem interferir de maneira direta no funcionamento do jogo para outros jogadores ativos no jogo, além dos aspectos da topologia do mundo virtual compartilhado;

- ❏ um método para definição de modelo de cenário de vídeo games *multiplayer* capaz de estimar a duração (mínima/média/máxima) de um jogo, considerando as interações/colaborações entre os diversos jogadores no mundo virtual;

- ❏ um método de prototipação de cenário de vídeo games baseado na ferramenta CPN Tools para o estudo de jogabilidade nas fases prévias do desenvolvimento de vídeo games nos casos *singleplayer* e *multiplayer*.

1.5 Organização do Documento de Tese

O presente documento encontra-se divido em 6 capítulos organizados da seguinte forma:

- ❏ o capítulo 2 apresenta os conceitos básicos necessários para o entendimento dos métodos propostos por esta pesquisa. Os conceitos sobre redes de Petri, propriedades de redes de Petri, WorkFlow nets, propriedade *soundness* de uma workflow net, WorkFlow nets Possibilísticas, redes de Petri Coloridas e utilização do CPN Tools são apresentados neste capítulo;

- ❏ o capítulo 3 tem como objetivo apresentar a literatura correlata a este trabalho. Neste capítulo são detalhadas as abordagens propostas por diferentes autores, bem como uma discussão sobre os aspectos abordados e as diferenças entre a literatura e a presente pesquisa;

- ❏ o capítulo 4 apresenta o método para a modelagem e análise de vídeo games no caso singleplayer usando o conceito de temporização das redes de Petri. Neste capítulo são definidos formalmente os modelos temporizados para a representação dos cenários, e também o método para realizar a análise do modelo por meio de replicações de simulações. É também apresentado um exemplo ilustrativo da aplicação do método proposto no capítulo;

- o capítulo 5 apresenta o método para a implementação de modelos de cenários *multiplayer* baseados nas WorkFlow nets Possibilísticas e grafo de estados para representar formalmente as interações entre os jogadores, o funcionamento das atividades do jogo e a topologia do mundo virtual. Os modelos temporizados para o caso multiplayer também são apresentados. Por fim, esse capítulo apresenta um método de análise da jogabilidade para vídeo games multiplayer. Para tanto é abordado um exemplo ilustrativo da aplicação do método proposto no capítulo;
- o capítulo 6 apresenta as considerações finais deste trabalho por meio das principais contribuições, as atividades futuras e as contribuições bibliográficas;
- por fim, as referências bibliográficas utilizadas ao longo deste trabalho são listadas.

Capítulo 2

Fundamentação Teórica

Este capítulo apresenta os conceitos necessários para o entendimento desta pesquisa. A seção 2.1 apresenta os conceitos de redes de Petri e suas propriedades. A seção 2.2 apresenta o conceito de um tipo particular de rede de Petri denominado de WorkFlow net. A seção 2.3 apresenta as definições e os conceitos das WorkFlow nets Possibilísticas. Já na seção 2.4 é apresentado um tipo específico de rede de Petri denominado de grafo de estados. Por fim, a seção 2.5 apresenta as redes de Petri Coloridas e o software CPN Tools.

2.1 Redes de Petri

A teoria das redes de Petri foi proposta inicialmente por Petri (1962) em sua tese intitulada "Comunicação com Autômatos". As redes de Petri sempre se mostraram muito úteis na representação de sistemas a eventos discretos permitindo a representação explícita de paralelismo e sincronização entre processos distintos e da concorrência para recursos compartilhados entre diversas aplicações (DAVID; ALLA, 2010). De acordo com Murata (1989), as redes de Petri são consideradas como uma ferramenta gráfica e matemática de representação formal que permite a modelagem, análise e controle de sistemas a eventos discretos que comportam atividades paralelas, concorrentes e assíncronas.

Segundo Murata (1989), uma rede de Petri pode ser vista como um tipo particular de grafo bipartido e direcionado com dois tipos de nós chamados de lugares e transições, onde os arcos conectam lugares a transições ou transições a lugares. Conexões entre dois nós do mesmo tipo não são permitidas. De acordo com David e Alla (2010), os elementos básicos que permitem a definição das redes de Petri são:

- **Lugar**: graficamente, um lugar é representado por um círculo. Esse elemento pode ser interpretado como uma condição, um estado parcial, um procedimento, a existência de um recurso, etc.;

- **Transição**: uma transição é representada graficamente por uma barra ou retângulo. Esse elemento é associado a um evento que ocorre no sistema;
- **Ficha** (*token*): graficamente, a ficha é representada por um ponto em um lugar. É um indicador significando que a condição associada ao lugar é verificada, como por exemplo, um recurso disponível em um certo processo.

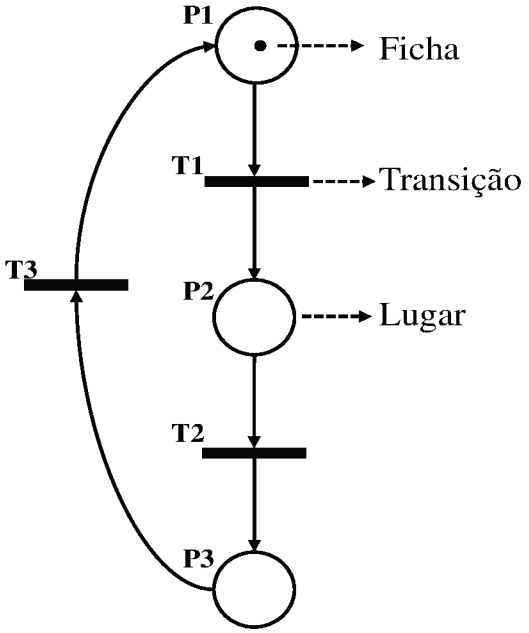


Figura 1 – Elementos básicos de uma rede de Petri.

A figura 1 ilustra um exemplo de uma rede de Petri e seus elementos básicos. Murata (1989) e Aalst e Stahl (2011) definem formalmente as redes de Petri da seguinte forma:

Definição 1 (Rede de Petri) *Uma rede de Petri é uma tripla* $PN = (P, T, F)$*, onde:*

- $\boldsymbol{P}$ *é um conjunto finito de lugares de PN.*
- $\boldsymbol{T}$ *é um conjunto finito de transições de PN.*
- $\boldsymbol{F} \subset (\boldsymbol{P} \times \boldsymbol{T}) \cup (\boldsymbol{T} \times \boldsymbol{P})$ *representa um conjunto de arcos direcionado que conectam lugares a transições e transições a lugares (relação de fluxo).*

De acordo com Aalst e Stahl (2011), os conceitos de lugares de entrada e lugares de saída são definidos em termos da relação de fluxo $\boldsymbol{F}$ da seguinte forma:

- Um lugar p é um lugar de entrada de uma transição t se $(p, t) \in$ F. O conjunto pré 't = {p | (p,t) $\in$ F} define todos os lugares de entrada de uma transição t.
- Uma lugar p é um lugar de saída de uma transição t se $(t, p) \in$ F. O conjunto pós t' = {p | (t,p) $\in$ F} define todos os lugares de saída de uma transição t.

Para ilustrar o conceito de lugares de entrada e saída, considere o exemplo apresentado na figura 1. O lugar *P1* é lugar de entrada da transição *T1*, pois existe um arco direcionado de *P1* para *T1*. Já o lugar *P3* é lugar de saída da transição *T2*, pois existe um arco direcionado de *T2* para *P3*.

Em um dado momento em uma rede de Petri, um lugar p pode conter zero ou mais fichas. Por exemplo, considerando o exemplo da figura 1, o lugar *P1* possui uma ficha, enquanto que o lugar *P2* possui zero fichas. A marcação de uma rede de Petri, denotada por M, diz respeito a distribuição de fichas nos lugares. Durante a execução da rede, o número de fichas pode mudar.

As transições são os componentes ativos que mudam a marcação da rede de acordo com os disparos. O estado ou marcação em uma rede de Petri é alterado de acordo com as seguintes regras de disparo (MURATA, 1989):

1. uma transição t é dita habilitada se cada lugar de entrada p de t contém pelo menos $w(p,t)$ fichas, onde $w(p,t)$ é o peso do arco de p para t;

2. uma transição habilitada pode ou não ser disparada, dependendo se o evento correspondente ocorre ou não;

3. um disparo de uma transição habilitada t remove *w(p,t)* fichas de cada local de entrada p de t e adiciona $w(t,p)$ fichas a cada lugar de saída p de t, onde $w(t,p)$ é o peso do arco de t para p.

Sendo assim, a ocorrência de um evento no sistema é representada pelo disparo da transição ao qual este evento está associado. Retirar as fichas de um lugar indica que esta condição não é mais verdadeira após a ocorrência do evento. Por outro lado, depositar fichas em um lugar indica que a atividade associada ao lugar está sendo executada após a ocorrência do evento. Deste modo, no momento em que um evento ocorre, a rede passa de um estado discreto para outro, alterando sua marcação. Assim, o comportamento dinâmico do sistema é traduzido pelo comportamento da rede de Petri (DAVID; ALLA, 2010).

Um exemplo do funcionamento de uma rede de Petri pode ser visto na figura 2. Na rede de Petri da figura 2, o conjunto de lugares é dado por P = {*P1*, *P2*, *P3*} e o conjunto de transições é dado por T = {*A*, *B*, *C*, *D*}. As transições *A* e *C* possuem o mesmo lugar de entrada, denotados por 'A = {*P2*} e 'C = {*P2*}, respectivamente. O lugar *P2* também é lugar de saída das transições *B* e *D*, denotados respectivamente por B' = {*P2*} e D' = {*P2*}. P1 é lugar de saída da transição A (A' = {*P1*}) e lugar de entrada da transição B ('B = {*P1*}). Enquanto que *P3* é lugar de saída da transição C (C' = {*P3*}) e lugar de entrada da transição D ('D = {*P3*}). O peso do arco que liga *P2* a C é três, isso significa que a transição C só poderá ser disparada se seu lugar de entrada, no caso *P2*,

possuir três fichas. O arco que liga D a *P2* também possui peso três. Assim, o disparo da transição D produzirá três fichas no lugar *P2*.

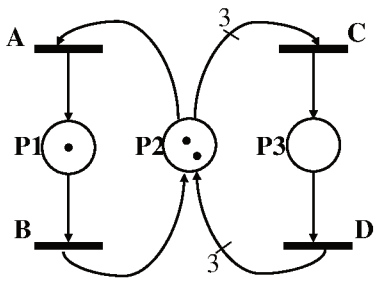


Figura 2 – Exemplo de uma rede de Petri.

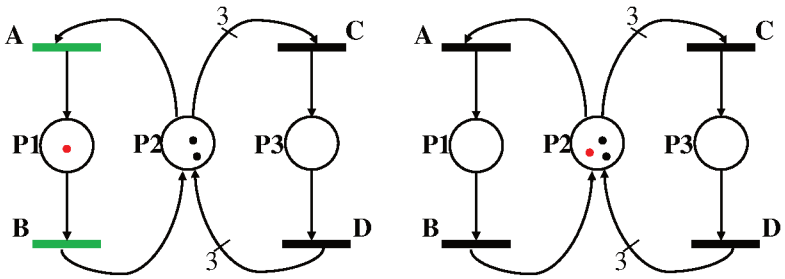
A figura 3 ilustra o funcionamento da rede de Petri. No início, o lugar *P2* possui duas fichas e o lugar *P1* possui uma ficha. Dessa forma, as transições A e B estão habilitadas, como mostra a figura 3(a). Considere que a transição B será disparada, produzindo assim uma ficha no lugar *P2* (figura 3(b)). Em seguida, a transição C estará apta para ser disparada (figura 3(c)). Ao ser disparada, a transição C consome três fichas de seu lugar de entrada e produz uma ficha em seu lugar de saída, como ilustra a figura 3(d). Após o disparo da transição C, a transição D poderá ser disparada, produzindo então três fichas em seu lugar de saída (figura 3(e)).

É importante destacar que as interpretações dos lugares e fichas são variadas. Podem representar entidades físicas, como uma peça por exemplo, e também entidades abstratas como uma condição. Dessa forma, as redes de Petri permitem uma visão sintética do sistema a ser modelado e autoriza procedimentos de análise.

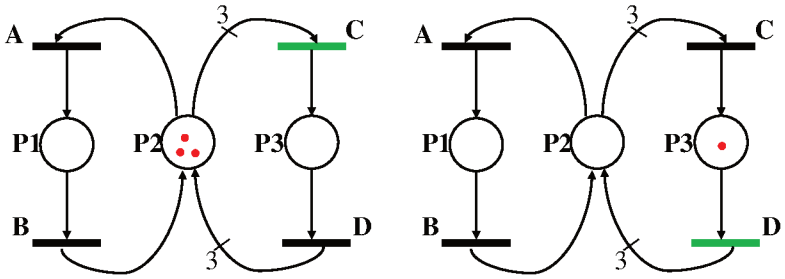
David e Alla (2010) afirmam que as redes de Petri não se resumem apenas a uma ferramenta que permite a modelagem de problemas que tenham atividades concorrentes, elas são utilizadas para descrever e analisar várias propriedades de um sistema por meio de diferentes métodos existentes (MURATA, 1989), (PETERSON, 1981).

A dinâmica de um sistema descrito por uma rede de Petri é dada pela evolução das marcações. A notação (PN, M) é usada para denotar uma rede de Petri PN com marcação inicial M. De acordo com o conjunto de marcações acessíveis a partir da marcação inicial (M_0), são definidas algumas propriedades comportamentais (reagrupadas sob o nome genérico de *boas propriedades* (CARDOSO; VALETTE, 1997)). Tais propriedades são definidas por Murata (1989) e apresentadas a seguir:

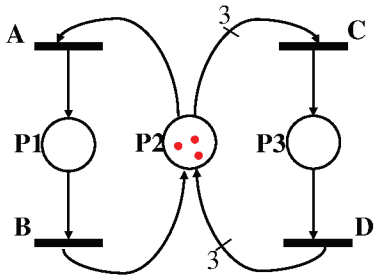
- **Alcançabilidade**: essa propriedade indica a possibilidade de alcançar uma determinada marcação. Uma marcação M_n é dita alcançável a partir da marcação M_0 se existe uma sequência finita de disparo de transições que conduza de M_n para M_0. Essa propriedade garante que certos estados serão alcançados ou não;



(a) Transições A e B habilitadas. (b) Após o disparo da transição B.



(c) Transição C habilitada. (d) Após o disparo da transição C.



(e) Após o disparo da transição D.

Figura 3 – Exemplo de funcionamento de uma rede de Petri.

- **Limitabilidade**: uma rede de Petri é dita limitada se o número de fichas em cada lugar da rede não exceder um inteiro positivo k para qualquer marcação alcançável a partir de M_0. Neste caso, a rede é dita k-limitada. Se a rede for limitada ao inteiro 1, então diz-se que ela é binária;

- **Vivacidade**: uma rede de Petri é dita viva se todas as suas transições são vivas. Uma transição t é dita viva se para cada marcação alcançável da rede, existe uma sequência de disparo S que sensibiliza t. Dessa forma, uma rede de Petri viva é uma rede onde todas as suas transições são disparáveis. Essa propriedade garante que o sistema é livre de *deadlock*;

- **Reiniciabilidade**: uma rede de Petri é dita reiniciável se, para qualquer marcação M, M_0 é alcançável a partir de M. Em outras palavras, a rede é reiniciável se é sempre possível voltar para a marcação inicial através de uma sequência de disparos,

seja qual for a marcação considerada.

Uma revisão completa sobre as redes de Petri pode ser encontrada em (MURATA, 1989), (CARDOSO; VALETTE, 1997) e (DAVID; ALLA, 2010).

2.2 WorkFlow nets

Um processo de negócio especifica quais tarefas precisam ser executadas e em qual ordem executá-las. Segundo Aalst (1998), processos de negócio são baseados em casos, ou seja, cada parte do trabalho é executada para um caso específico. Uma rede de Petri que modela uma processo de negócio é chamada de WorkFlow net (WF-net) (AALST, 1998), (AALST; HEE, 2004).

De acordo com Aalst (1998), uma WF-net satisfaz as seguintes propriedades:

1. uma WF-net tem apenas um lugar de início (i) e apenas um lugar de fim (o). Esses dois lugares são tratados como lugares especiais. O lugar i tem apenas arcos de saída e o lugar o tem apenas arcos de entrada;

2. uma ficha no lugar i representa um caso que precisa ser tratado. Uma ficha no lugar o representa um caso que já foi tratado;

3. em uma WF-net, cada tarefa (transição) e condição (lugar) deve estar em um caminho que se encontra entre o lugar de início (i) e o lugar de término (o).

A definição formal de uma WorkFlow net (AALST, 1998) é apresentada a seguir.

Definição 2 (WorkFlow net) *Uma rede de Petri PN = (P,T,F) é uma WorkFlow net se, e somente se:*

- ❑ *existe um único lugar de início* $\mathrm{i} \in \mathrm{P}$ *tal que* $\bullet i = \emptyset$*;*
- ❑ *existe um único lugar de fim* $\mathrm{o} \in \mathrm{P}$ *tal que* $o\bullet = \emptyset$*;*
- ❑ *todo lugar* $\mathrm{x} \in \mathrm{P} \cup \mathrm{T}$ *está em um caminho entre os lugares* i *e* o*.*

Uma WF-net tem apenas um lugar de início e um lugar de fim porque qualquer caso que for tratado pelo procedimento, representado por uma WF-net, é criado quando ele entra no sistema de gerenciamento de processo de negócio e é excluído assim que é completamente tratado pelo sistema. Em outras palavras, a WF-net especifica o ciclo de vida de um caso. A terceira propriedade da definição 2 trata de garantir que não existirá tarefas e/ou condições pendentes, ou seja, tarefas e condições que não contribuem para o processamento dos casos (AALST, 1998).

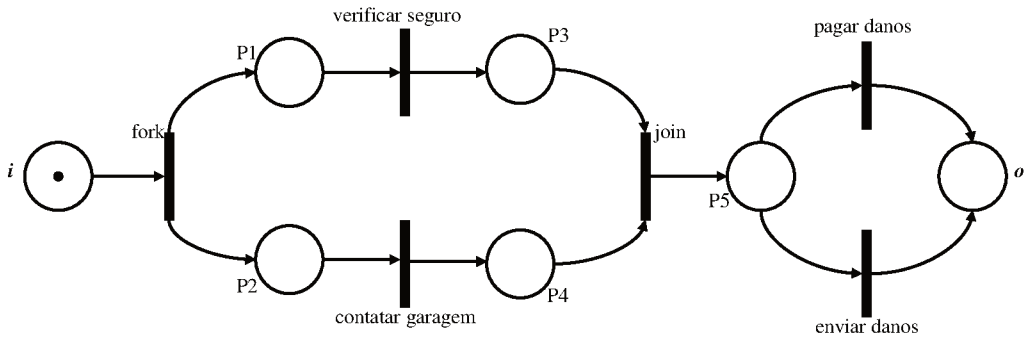


Figura 4 – Exemplo de uma WorkFlow net.

Modelar um processo de negócio em termos de uma WF-net é bem direto: as tarefas são modeladas por transições, condições são modeladas por lugares e os casos são modelados pelas fichas (AALST, 1998). Para exemplificar o mapeamento de um processo por meio de uma WF-net, considere o exemplo ilustrado na figura 4, apresentado em Aalst (1997), que trata do processamento de pedidos relacionados a danos de carros. As tarefas necessárias para processar um pedido são: *verificar seguro*, *contatar garagem*, *pagar danos* e *enviar carta*. As tarefas *verificar seguro* e *contatar garagem* determinam se o pedido é justificado, e podem ser executadas em qualquer ordem. Se o pedido é justificado, então o dano é pago (tarefa *pagar danos*). Caso contrário, uma carta de rejeição é enviada ao requerente (tarefa *enviar carta*).

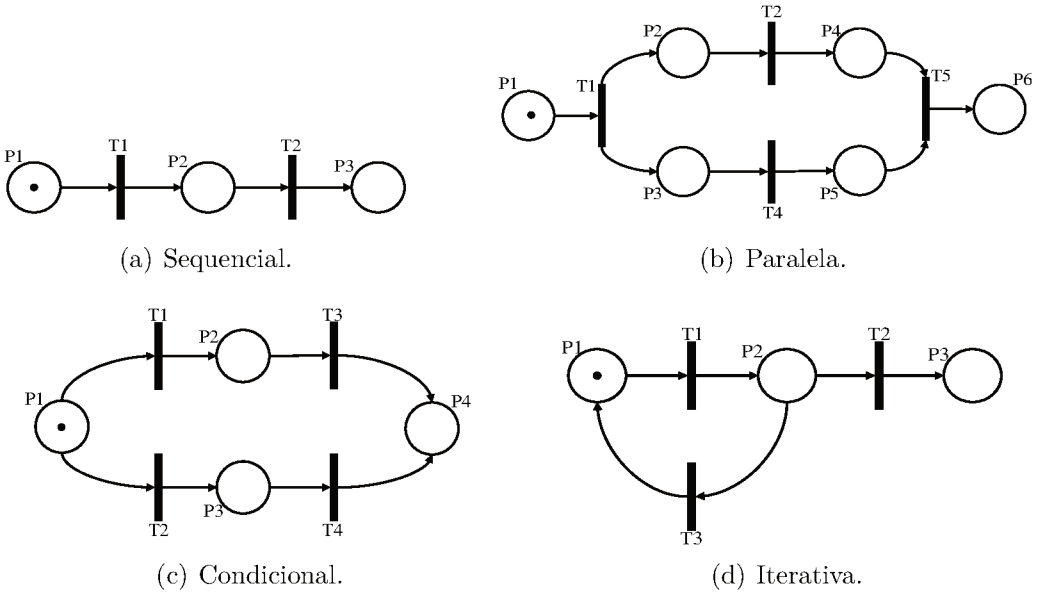
A ordem em que as tarefas são executadas varia de caso para caso. Por meio do roteiro de um caso ao longo de uma série de tarefas é possível determinar qual a ordem das tarefas que precisam ser executas. De acordo com Aalst e Hee (2004), quatro construções básicas são consideradas para o roteamento de tarefas, são elas:

1. **Sequencial**: refere-se a execução sequencial quando as tarefas precisam ser executadas uma após a outra. Quando uma tarefa precisa ser executada após uma outra, tem-se então uma dependência entre essas tarefas. A figura 5(a) ilustra um exemplo de roteamento sequencial;

2. **Paralela**: quando mais de uma tarefa pode ser executada simultaneamente ou em qualquer ordem. Ambas tarefas podem ser executadas sem que uma interfira no resultado da outra. Um exemplo desse tipo de roteamento pode ser visto na figura 5(b);

3. **Condicional**: quando pode existir uma escolha entre duas ou mais tarefas, como ilustrado na figura 5(c);

4. **Iterativa**: quando é necessário repetir uma mesma tarefa ou uma sequência de tarefas várias vezes. A figura 5(d) ilustra um exemplo de um roteamento iterativo.



(a) Sequencial. (b) Paralela.

(c) Condicional. (d) Iterativa.

Figura 5 – Construções básicas para o roteamento de tarefas em uma WF-net.

Na WF-net da figura 4, as tarefas *verificar seguro* e *contatar garagem* são um exemplo de roteamento paralelo. Enquanto que as tarefas *pagar dano* e *enviar carta* fazem parte de um roteamento condicional.

2.2.1 Propriedade Soundness

Soundness é um critério de corretude definido para as WorkFlow nets. Uma WorkFlow net é dita *sound* se, e somente se, os seguintes requisitos forem satisfeitos (AALST; HEE, 2004), (AALST, 1998):

1. para cada ficha colocada no lugar de início i, uma, e apenas uma ficha deve aparecer no lugar de término o;

2. quando uma ficha aparece no lugar o, todos os outros lugares estão vazios para o caso em questão;

3. para cada transição (tarefa), é possível evoluir da marcação inicial para uma marcação onde essa transição é sensibilizada, ou seja, em uma WorkFlow net não deve haver transição morta.

A propriedade *soundness* está relacionada com a dinâmica das WorkFlow nets. O primeiro requisito significa que todo caso será completado após um período de tempo. O segundo requisito significa que uma vez que um caso é completado, nenhuma referência a ele permanecerá no processo. Já o terceiro requisito afirma que todas as tarefas em uma WF-net podem ser, a princípio, executadas. A definição formal da propriedade *soundness*

no contexto de uma WF-net foi proposta em (AALST, 1998) e (AALST; HOFSTEDE, 2000).

Definição 3 (Soundness) *Um processo modelado por uma WF-net PN = (P, T, F) é dito sound se, e somente se:*

- *para cada marcação M alcançável a partir da marcação* i*, existe uma sequência de disparo que leva da marcação M para a marcação* o*. Formalmente:*

$$\forall_M (i \xrightarrow{*} M) \rightarrow (M \xrightarrow{*} o);$$

- *a marcação* o *é o único estado alcançável a partir da marcação* i *com pelo menos uma ficha no lugar* o*. Formalmente:*

$$\forall_M (i \xrightarrow{*} M \wedge M \geq o) \rightarrow (M = o);$$

- *não há transições mortas em* (PN, i)*. Formalmente:*

$$\forall_{t \in T} \exists_{M,M'} i \xrightarrow{*} M \xrightarrow{*} M'.$$

A propriedade *soundness* é um critério importante a ser satisfeito quando são considerados processos de negócios. No contexto das WF-nets, a prova desse critério está relacionada com um problema de análise qualitativa de um modelo. Em Aalst e Hee (2004), alguns métodos para provar a propriedade *soundness* são apresentados. Um desses métodos em específico será utilizado neste trabalho e apresentado na seção seguinte.

2.2.2 Algoritmo de Verificação da Propriedade Soundness

Em Aalst (1998) e Aalst e Hofstede (2000), um método foi proposto para a verificação da propriedade *soundness* de uma WorkFlow net. Este método traduz a propriedade *soundness* através de duas propriedades bem conhecidas: vivacidade (*liveness*) e limitabilidade (*boundedness*) (AALST; HEE, 2004).

Dado uma WorkFlow net PN = (P,T,F), deve-se decidir se PN é *sound*. Para tanto, define-se uma rede estendida $\overline{PN} = (\overline{P}, \overline{T}, \overline{F})$. $\overline{PN}$ é uma rede de Petri obtida adicionando-se uma transição extra t^*. A transição t^* conecta o lugar de término (o) ao lugar de início (i). A figura 6 ilustra a relação entre PN e $\overline{PN}$.

Para uma WorkFlow-net arbitrária PN e sua correspondente rede de Petri estendida $\overline{PN}$, o seguinte teorema (AALST, 1997) pode ser provado.

Teorema 1 (*Soundness*) *Uma WF-net é dita sound se, e somente se,* $(\overline{PN}, i)$ *é viva (live) e limitada (boundedness).*

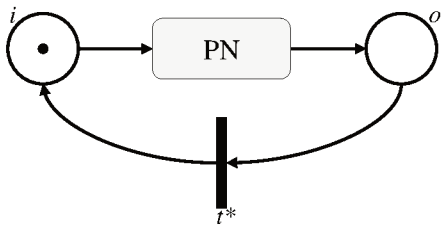


Figura 6 – Exemplo de uma rede de Petri estendida $\overline{PN}$.

A prova deste teorema pode ser encontrada em (AALST, 1997) e (AALST, 1996). Dessa forma, a verificação da propriedade *soundness*, em uma WF-net, resume-se em verificar se a rede de Petri estendida $\overline{PN}$ é viva e limitada. Isso significa que ferramentas de análise baseadas em redes de Petri podem ser utilizadas para definir o critério *soundness* da rede (AALST, 1996), (AALST, 1997), (AALST, 1998).

2.3 WorkFlow net Possibilística

De acordo com Rezende, Julia e Cardoso (2012), a interação do comportamento humano no gerenciamento dos processos pode introduzir algumas incertezas na execução dos processos. Quando incertezas são inseridas no modelo, a rede de Petri Clássica pode não ser suficiente para lidar com o comportamento do modelo. Assim, Rezende, Julia e Cardoso (2012) apresentaram um modelo que combina a estrutura de roteamento das WorkFlow nets com a marcação e o disparo impreciso das redes de Petri Possibilísticas. Esse tipo de modelo é apresentado pelos autores como WorkFlow net Possibilística. A definição de uma WorkFlow net Possibilística ((REZENDE; JULIA; CARDOSO, 2012), (REZENDE.; JULIA.; CARDOSO., 2016)) é dada como se segue.

Definição 4 (WorkFlow net Possibilística (WF-net Possibilística)) *Uma WorkFlow net Possibilística pode ser definida pela tupla* $R = <P, T, C_{aso}, V, Pré, Pós, A_{tc}, A_{ta}, M_0>$ *onde:*

- $\boldsymbol{C_{aso}}$ *representa a classe de objeto "Caso", onde um conjunto de atributos é definido e eventualmente organizado em uma hierarquia;*
- ***P*** *é um conjunto finito de lugares, todos do tipo "Caso"(incluindo o lugar de entrada de Início, o lugar de saída de Término e os lugares envolvidos nos diversos roteiros da WorkFlow net);*
- ***T*** *é um conjunto finito de transições;*
- ***V*** *é um conjunto de variáveis formais do tipo "Caso";*
- ***Pré*** *é a função lugar precedente que a cada arco de entrada de uma transição* $t \in T$ *faz corresponder a uma soma formal de elementos de V;*

- ***Pós*** *é a função lugar seguinte que a cada arco de saída de uma transição $t \in T$ faz corresponder a uma soma formal de elementos de V;*
- $\boldsymbol{A_{tc}}$ *é uma aplicação que, para cada transição $t \in T$, associa uma função de autorização que envolve o conjunto de atributos e métodos dos objetos por meio das variáveis formais V associados aos arcos de entrada;*
- $\boldsymbol{A_{ta}}$ *é uma aplicação que, para cada transição $t \in T$, associa uma ação que envolve os atributos ou métodos das variáveis formais associados aos arcos de entrada permitindo modificar seus atributos específicos por meio da invocação de seus métodos;*
- $\boldsymbol{M_0}$ *é a marcação inicial que associa, ao lugar de Início, uma soma formal dos objetos do tipo "Caso"(n-uplas de instâncias da classe "Caso").*

O modelo apresentado em Rezende, Julia e Cardoso (2012) também insere a noção e incerteza que exprime o fato de que a existência de um objeto é conhecida, mas a sua localização é imprecisa. A partir disso, a marcação da WF-net possibilística pode ser precisa ou imprecisa permitindo assim a existência de dois tipos de disparos: certo ou incerto (pseudo-disparo). As definições desses conceitos são apresentadas a seguir.

Definição 5 (Tipos de Marcação) *A localização dos objetos em uma WF-net Possibilística permite a representação de uma:*

- ***Marcação precisa****: cada objeto do tipo "Caso"está localizado em apenas um lugar, ou seja, se a possibilidade do objeto* b *estar no lugar* p *é 1 ($\pi_b(p) = 1$), então a possibilidade do objeto* b *estar nos outros lugares do modelo diferentes de* p *é 0 ($\forall p_i \neq p$, $\pi_b(p) = 0$);*
- ***Marcação imprecisa****: um mesmo objeto do tipo Caso está em dois ou mais lugares com possibilidade igual a 1 ($\pi_b(p) = 1$ e $\exists p_j \neq p_i \mid \pi_b(p_j) = 1$).*

Definição 6 (Tipos de Disparo) *O disparo de uma transição $t \in T$ em uma WF-net possibilística pode ser de dois tipos:*

- ***Disparo certo****: corresponde ao disparo de uma transição conforme a definição original da rede de Petri. Assim sendo, o término do disparo, neste caso a finalização, coincide com o início do mesmo. Neste tipo de disparo a localização de todas as instâncias dos objetos envolvidos deve ser precisa. Como consequência, a nova marcação do sistema também é precisa e obtida a partir da remoção das instâncias dos objetos do tipo Caso dos lugares de entrada de t e da adição de novas instâncias de objetos do tipo Caso nos lugares de saída de t;*

- ***Disparo Incerto (ou pseudo-disparo)****: neste tipo de disparo considera-se apenas o início do disparo uma vez que não existe informação suficiente para confirmar se os eventos esperados associados à transição realmente ocorreram ou não, ou seja, as instâncias dos objetos do tipo Caso são adicionados aos lugares de saída da transição mas não são removidos dos lugares de entrada da mesma.*

Definição 7 (Função de Autorização) *A interpretação da WF net Possibilística é dada pela função de autorização definida por:*

$$\eta_{x_1,...,x_n}: \ T \rightarrow \{Falsa, Incerta, Verdadeira\}$$

onde $x_1, ..., x_n$ *são as variáveis associadas aos arcos de entrada da transição* t.

Definição 8 (Regras de Disparo) *Seja t uma transição e* $s_1, ..., s_n$ *uma possível substituição dos valores de atributos dos objetos* $b_1, ..., b_n$ *nas variáveis* $x_1, ..., x_n$ *para disparar t, as regras de disparo são definidas da seguinte forma:*

- *se a transição não está habilitada, mas sua interpretação é verdadeira, então um alarme é ativado dado que isto corresponde a uma situação proibida;*
- *se a transição está habilitada por uma marcação precisa e sua interpretação é verdadeira, então a transição é disparada com certeza, isto é, as instâncias dos objetos são removidos dos lugares de entrada da transição e a ação associada (disparo certo, com marcação precisa e* $\eta(t) = verdadeira$*) é então executada e novas instâncias dos objetos são produzidas nos lugares de saída da transição;*
- *se a transição está habilitada por uma marcação precisa ou imprecisa e sua interpretação é incerta, então a transição é disparada com incerteza, isto é, as instâncias dos objetos não são removidos dos lugares de entrada da transição, a ação associada (disparo incerto, com* $\eta(t) = incerta$*) é executada e novas instâncias dos objetos são produzidas nos lugares de saída da transição;*
- *se a transição está habilitada por uma marcação imprecisa e sua interpretação é verdadeira, então um algoritmo de defuzzificação é chamado para computar uma nova distribuição de possibilidade para as instâncias dos objetos pertencentes à rede. Tal algoritmo é responsável por finalizar ou cancelar os disparos incertos relacionados à transição. Neste caso, quando um disparo incerto de uma transição é finalizado, a ação correspondente ao disparo incerto finalizado é executada e, quando é cancelado, a ação correspondente ao disparo incerto cancelado que é executada. Após a execução do algoritmo, se a transição estiver habilitada por uma marcação precisa, então será disparada com certeza.*

Modelos baseados em redes de Petri podem ser diretamente executados usando um mecanismo de inferência conhecido como algoritmo do "jogador" (*Token Player*) que permite um monitoramento simplificado dos processos representados. A figura 7 ilustra um algoritmo do "jogador" para uma rede de Petri Clássica representado por um diagrama de atividades. Esse algoritmo é baseado apenas em eventos esperados.

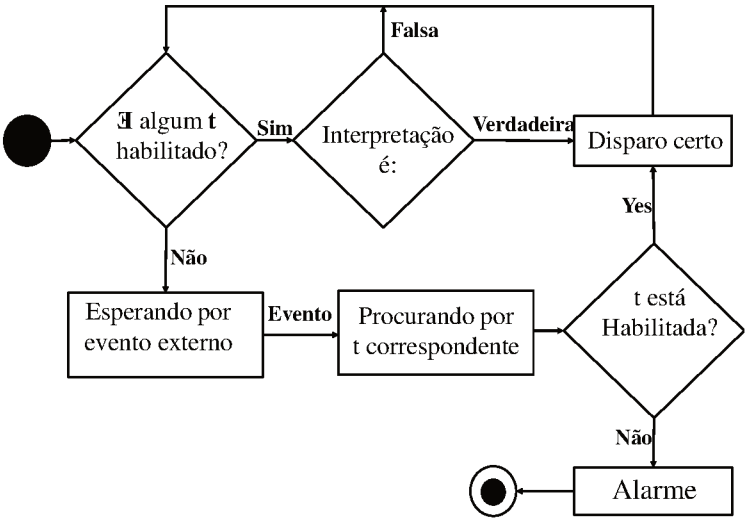


Figura 7 – Algoritmo do jogador de uma rede de Petri Clássica.

Em sistemas complexos, eventos inesperados podem acontecer e inserir um certo grau de incerteza que precisa ser levada em consideração. Assim, quando um evento inesperado precisa ser levado em consideração no processo, o algoritmo do jogador ilustrado na figura 8 (e apresentado em (REZENDE.; JULIA.; CARDOSO., 2016)) é considerado.

O restabelecimento do estado certo no sistema ocorrerá quando a função de autorização de uma transição t (η(t)) for verdadeira e o posicionamento dos objetos nos lugares relacionados à ela for incerta (REZENDE.; JULIA.; CARDOSO., 2016). Este algoritmo recupera apenas o estado certo dos objetos envolvidos neste evento, ou seja, caso existam outros pseudo-disparos que não se relacionem direta ou indiretamente a este evento os mesmos não serão recuperados, continuando assim numa situação de imprecisão do conhecimento (REZENDE; JULIA; CARDOSO, 2012).

O pseudo-disparo de uma transição t pode ser considerado apenas como o início de um disparo normal. As fichas são adicionadas nos lugares de saída, mas não são removidas dos seus lugares de entrada. Isso ocorre pela falta de certeza se a transição t foi ou não realmente disparada. Ao realizar o novo cálculo das distribuições de possibilidade, deve-se decidir, para cada transição que foi pseudo-disparada, se as mesmas foram ou não efetivamente disparadas. Se as transições foram disparadas, considera-se que o disparo foi arquivado (t realmente foi disparada) e, caso o contrário, o disparo foi cancelado (t não foi disparada).

De acordo com Rezende, Julia e Cardoso (2012), para que o procedimento de recu-

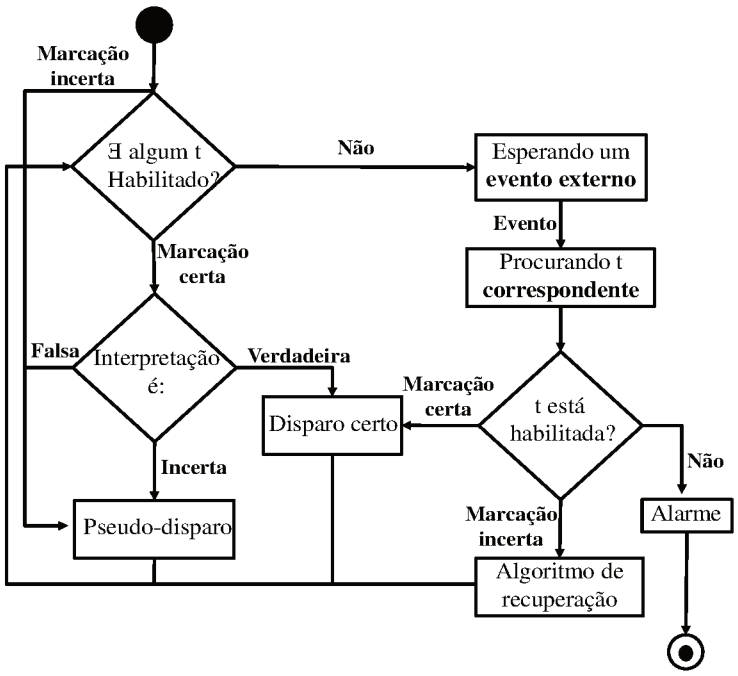


Figura 8 – Algoritmo do jogador de uma rede de Petri Possibilística.

peração funcione, nenhum disparo incerto poderá ser realizado caso a distribuição de possibilidade dos objetos pertencentes aos lugares de saída da transição seja igual à 1. Também é preciso considerar que uma transição não pode ser pseudo-disparada, em um dado momento, por mais de uma vez. Assim, o algoritmo de deffuzificação (CARDOSO; VALETTE; DUBOIS, 1991) é descrito como se segue:

1. assume-se o caso em que uma mensagem chega e $\eta(t)$ = verdadeira. Se a transição t estiver em processo de pseudo-disparo (anteriormente com η(t) = incerta), então cancelar esse disparo;

2. coloca-se na lista LP os lugares de entrada e saída de t;

3. se LP estiver vazia, executar o passo 6, caso contrário assumir o lugar p como o primeiro elemento de LP e remover p de LP;

4. se existir uma transição de entrada t' de p diferente de t em processo de pseudo-disparo, então arquivar este disparo, colocar os lugares de entrada e saída de t' (que são diferentes de p) na lista LP e executar o passo 4; caso contrário, executar o passo 5;

5. se existir uma transição de saída t' de p diferente de t em processo de pseudo-disparo, então cancelar este disparo, colocar os lugares de entrada e saída que são diferentes de p na lista LP e executar o passo 5; caso contrário, executar o passo 3;

6. o disparo normal da transição t pode ser executado, pois a localização dos objetos é certa.

Rezende, Julia e Cardoso (2012), afirmam que o algoritmo de deffuzificação sempre para, pois o número de lugares e transição é finito e por assumir que não existe um ciclo de possíveis lugares para um objeto.

2.4 Grafos de estado

Uma rede de Petri não marcada é um grafo de estado se, e somente se, cada transição tiver exatamente um arco de entrada e um arco de saída (DAVID; ALLA, 1994), como ilustrado na figura 9. Desde que cada transição possua apenas uma entrada e uma saída lugar, a representação de um gráfico de estados no sentido clássico (ou diagrama de estados, ilustrado na figura 10) contém as mesmas informações (DAVID; ALLA, 2010).

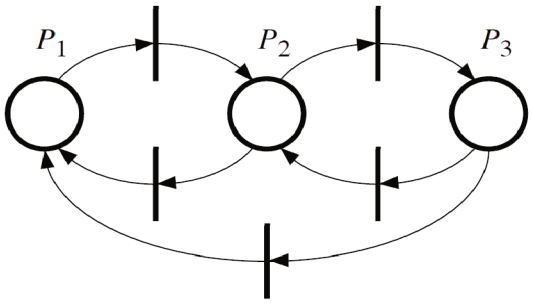


Figura 9 – Exemplo de um grafo de estado não marcado.

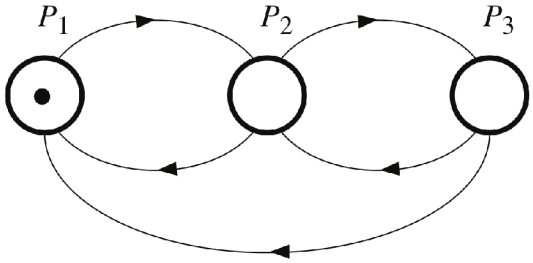


Figura 10 – Exemplo de um diagrama de estado.

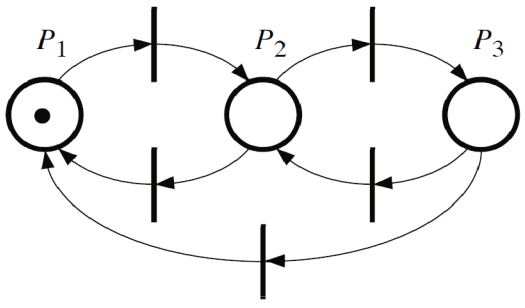


Figura 11 – Exemplo de um grafo de estado marcado com apenas uma ficha.

Uma rede de Petri marcada conhecida como grafo de estado, será equivalente a um grafo de estado no sentido clássico (representando um autômato que está em apenas um estado por vez) se, e somente se, contém exatamente uma ficha localizada em um dos lugares da rede. Na figura 11 é apresentado um exemplo de um grafo de estado com uma ficha. Em um grafo de estado, o peso de todos os arcos é igual a 1 (DAVID; ALLA, 2010).

2.5 Redes de Petri Coloridas

Redes de Petri são tradicionalmente divididas em redes de Petri de alto nível e redes de Petri de baixo nível. Redes de Petri de baixo nível são caracterizadas por possuírem fichas simples (números naturais associados aos lugares) que geralmente indicam uma condição ativa num sistema ou a existência de um recurso. Já as redes de Petri de alto nível se preocupam com casos reais, pois permitem a construção de modelos compactos e parametrizados.

Redes de Petri clássicas pertencem à classe das redes de Petri de baixo nível. Elas permitem a representação de paralelismo e sincronização. Dessa forma, são apropriadas para a modelagem de sistemas distribuídos. No entanto, para a modelagem de sistemas complexos ou com um comportamento temporal, o modelo clássico das redes de Petri é insuficiente (AALST; STAHL, 2011). Nesse sentido, muitas extensões do modelo básico da rede de Petri surgiram da necessidade de representar com precisão os sistemas complexos. Uma dessas extensões foi a rede de Petri Colorida (*Colored Petri Net* - CP-net).

A ideia por trás das redes de Petri Coloridas é reunir a capacidade de representação da sincronização e do paralelismo das redes de Petri com o poder expressivo das linguagens de programação com seus tipos de dados e o conceito de tempo. As CP-nets pertencem à classe das redes de Petri de alto nível e são caracterizadas pela combinação das redes de Petri e da linguagem de programação funcional CPN ML (MILNER et al., 1997). Assim, o formalismo das rede de Petri é adequado para descrever ações concorrentes e síncronas, enquanto que a linguagem de programação funcional pode ser usada para definir tipos de dados específicos e manipulá-los (JENSEN; KRISTENSEN, 2009).

A definição formal das redes de Petri Coloridas é dada por Jensen e Kristensen (2009) e é apresentada como se segue.

Definição 9 (Rede de Petri Colorida) *Uma rede de Petri Colorida não hierárquica (CPN) é uma tupla* $CPN = (P, T, A, \Sigma, V, C, G, E, I)$ *onde:*

- $\boldsymbol{P}$ *é o conjunto finito de lugares;*
- $\boldsymbol{T}$ *é o conjunto finito de transições* T *tal que* $P \cap T = \emptyset$*;*
- $\boldsymbol{A} \subseteq P \times T \cup T \times P$ *é o conjunto de arcos direcionados;*

- Σ *é um conjunto finito de conjunto de cores (color sets) não vazios;*
- ***V*** *é um conjunto finito de variáveis tipadas tal que* $Tipo[v] \in \Sigma$ *para toda variável* $v \in V$*;*
- ***C****:* $P \rightarrow \Sigma$ *é a função do conjunto de cores que associa a cada lugar um conjunto de cor;*
- ***G****:* $T \rightarrow EXPR_v$ *é uma função de guarda (expressão booleana) que associa uma guarda a cada transição t tal que* $Tipo[G(t)] = Bool$*;*
- ***E****:* $A \rightarrow EXPR_v$ *é uma função de expressão de arco que associa uma expressão a cada arco tal que* $Tipo[E(a)] = C(p)_{MS}$*, onde p é o lugar conectado ao arco a;*
- ***I****:* $P \rightarrow EXPR$ *é uma função de inicialização que atribui uma expressão de inicialização a cada lugar p tal que* $Type[I(p)] = C(p)_{MS}$*.*

Os estados de uma CP-net são representados pela marcação dos lugares. Cada lugar tem um tipo associado que determina o tipo de dado que o lugar pode conter. Cada lugar conterá um número variado de fichas. Por sua vez, cada ficha tem um valor, denominado de cor no contexto das CP-nets, que pertence ao tipo de dado associado com o lugar (AALST; STAHL, 2011). Essas cores não representam apenas cores ou padrões, elas podem representar desde dados primitivos (número inteiros, por exemplo) até tipos de dados complexos (produto cartesiano de valores ou ainda uma estrutura de dados, por exemplo) (JENSEN; KRISTENSEN, 2009). A figura 12 ilustra um exemplo dos elementos básicos de uma CP-net.

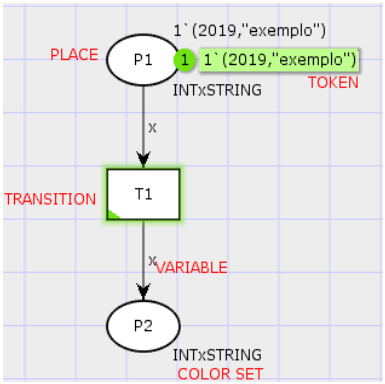


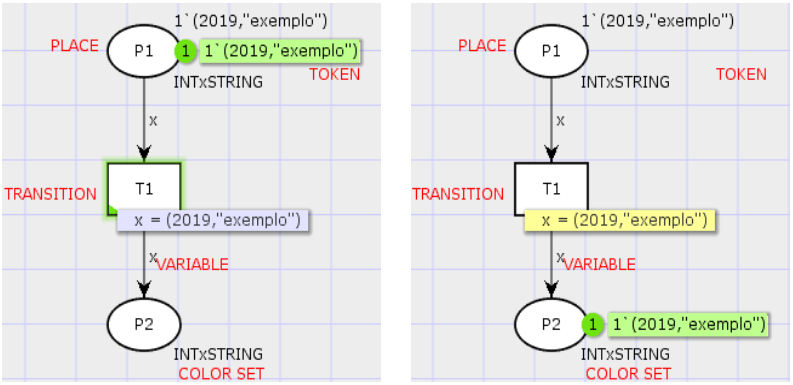
Figura 12 – Elementos básicos de uma rede de Petri Colorida.

A CP-net da figura 12 possui dois lugares denominados *P1* e *P2*. Os lugares possuem o tipo de dado (*color set*) *INTxSTRING*, assim como a variável x do mesmo tipo associada aos arcos da rede. Este tipo de dado é formado pelo produto cartesiano de dois tipos: *INT* (inteiros) e *STRING* (cadeia de caracteres). Os lugares da rede aceitam apenas fichas do mesmo tipo, ou seja, fichas do tipo *INTxSTRING*. Nesse sentido, a inscrição

1'(2019, "exemplo") corresponde a uma ficha (1‘) cujos atributos são dados pelo valor inteiro (2019) e pela *string ("exemplo")*.

As declarações dos *color sets* utilizados, bem como das variáveis, são indispensáveis para a criação de um modelo CP-net. O software CPN Tools possui um ambiente em que integra a área para criação gráfica da rede como também a área para a declaração dos tipos de dados das variáveis e das funções, entre outros. Para o exemplo da figura 12, foi necessário declarar três *color sets* e uma variável da seguinte forma:

```
colset INT = int;
colset STRING = string;
colset INTxSTRING = product INT * STRING;
var x: INTxSTRING;
```



(a) Disparo da transição T1 e associação do valor a variável *x*.

(b) Após o disparo da transição T1.

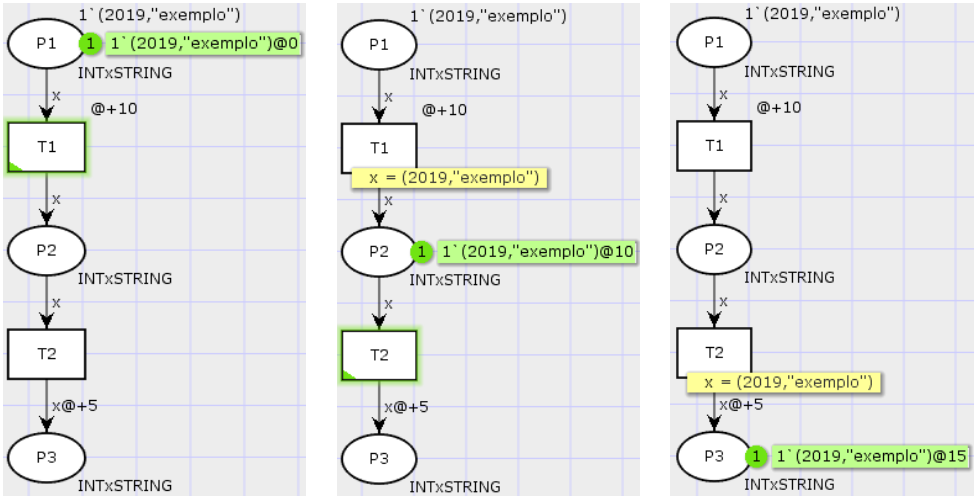
Figura 13 – Exemplo de funcionamento de uma rede de Petri Colorida.

No exemplo da figura 13, a transição T1 será habilitada apenas se em seu lugar de entrada (*P1*) existir pelo menos uma ficha. Durante o disparo de uma transição no modelo, as variáveis dos arcos de entrada serão substituídas pelo valor da ficha. Para exemplificar, considere o exemplo da figura 13(a) que ilustra o disparo da transição T1. No momento do disparo o valor *(2019,"exemplo")* é associado com a variável *x*. Após disparar, T1 produzirá uma ficha em P2, como ilustra a figura 13(b).

As redes de Petri Coloridas também permitem adicionar informações de tempo nos modelos a fim de investigar propriedades relativas a performance do sistema modelado. Para isso, foi introduzido um relógio global que representa o tempo do modelo. De acordo com Kristensen, Christensen e Jensen (1998), os valores do relógio podem ser discretos ou contínuos.

Em um modelo CP-net temporizado, um valor de tempo é associado a ficha, conhecido como *timestamp* (JENSEN; KRISTENSEN, 2009). Para calcular o *timestamp* associado a uma ficha, é necessário usar uma inscrição de atraso associada à transição ou aos arcos de

saída das transições. Quando uma inscrição/função de tempo é associada a uma transição, um tempo é adicionado em todas as fichas produzidas por essa transição. Por outro lado, quando uma inscrição/função de tempo é associada aos arcos de saída de uma transição, um tempo é adicionado apenas nas fichas criadas nesse arco (JENSEN; KRISTENSEN, 2009).



(a) Rede de Petri Colorida Temporizada.

(b) Disparo da transição T1 com atraso de tempo em 10 unidades.

(c) Disparo da transição T2 com o acréscimo de tempo em 5 unidades no arco de saída.

Figura 14 – Exemplo de funcionamento de uma rede de Petri Colorida Temporizada.

Para exemplificar o funcionamento de uma CP-net temporizada, considere o modelo apresentado na figura 14. A transição T1 representa uma operação que demora 10 unidades de tempo para ser concluída. Para tanto, uma inscrição de tempo $@+10$ foi associada a T1. Assim, toda vez que T1 for disparada, serão acrescentados 10 unidades de tempo nas fichas produzidas por T1. A figura 14(b) ilustra a rede após o disparo da transição T1. Note que antes do disparo de T1, o tempo associado à ficha é de @0 (figura 14(a)). Após o primeiro disparo de T1, o tempo passa a ser *@10*, como mostra a figura 14(b). O arco de saída da transição T2 possui a inscrição $@+5$. Assim, o *timestamp* das fichas produzidas nesse arco após o disparo da transição T2 será a soma do valor do relógio global com a quantidade de unidades de tempo especificada pela inscrição do arco, como pode ser visto na figura 14(c).

Para adicionar a característica de tempo nas fichas é necessário que a declaração do *color set* seja feita de maneira adequada adicionando a palavra reservada *timed* ao final da declaração. Porém, as declarações de variáveis e fichas continuam da mesma forma que um modelo não-temporizado. Para o exemplo da figura 14, as declarações dos *color sets* utilizados são dados da seguinte forma:

```
colset INT = int;
```

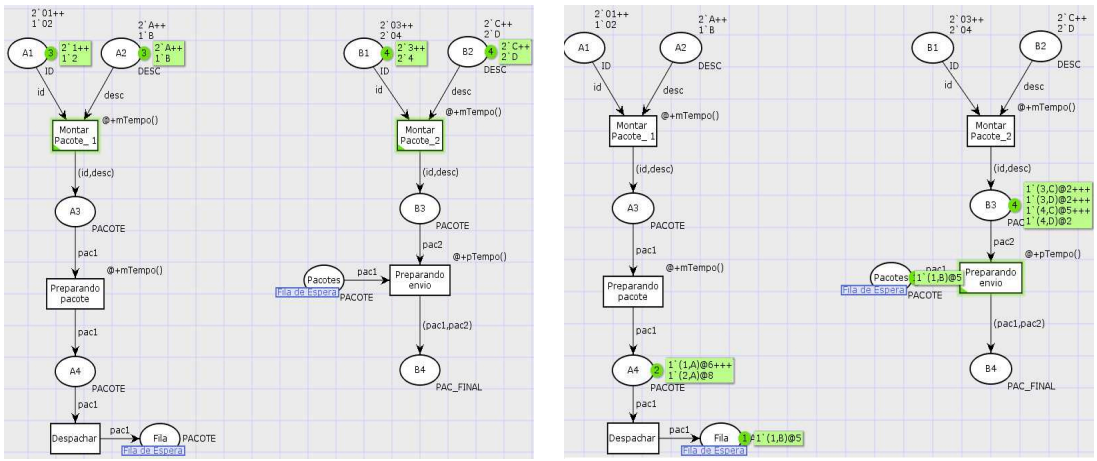
```
colset STRING = string;
colset INTxSTRING = product INT * STRING timed;
var x: INTxSTRING;
```

Uma característica interessante da implementação das redes de Petri Coloridas por meio da ferramenta CPN Tools é a possibilidade de estruturar os modelos em diferentes módulos. O conceito de módulos em CP-nets é baseado em um mecanismo de estruturação hierárquica. A ideia básica da hierarquia por trás das CP-nets é permitir a construção de um amplo modelo combinando um número de pequenas redes de Petri Coloridas em um único modelo (JENSEN; KRISTENSEN, 2009). De acordo com Aalst e Stahl (2011), essa característica facilita a modelagem de sistemas grandes e complexos, tais como sistemas de informação e de processos de negócio.

A hierarquia das redes de Petri Coloridas oferece um conceito conhecido como *fusion places* (lugares de fusão). Esse conceito permite especificar um conjunto de lugares que são considerados idênticos (KRISTENSEN; CHRISTENSEN; JENSEN, 1998). Isto significa que todos esses lugares representam um único lugar conceitual, ainda que sejam desenhados como vários lugares individuais. Esses lugares são chamados, individualmente, *fusion places*, e um conjunto de *fusion places* é chamado de *fusion set* (conjunto de fusão). Qualquer ação que acontecer a um lugar do conjunto de fusão, também acontece aos outros lugares do mesmo conjunto. Assim, se uma ficha for adicionada/consumida de um dos lugares, uma ficha idêntica também será adicionada/consumida em todos os outros lugares do conjunto de fusão.

Para exemplificar o funcionamento de *fusion places*, considere o exemplo da figura 15. A figura 15(a) representa dois modelos, distintos, que produzem pacotes. Cada pacote possui um número identificador e uma descrição. Os lugares de *A1* até *A4* pertencem ao modelo 1, e os lugares de *B1* até *B4* pertencem ao modelo 2. Ambos modelos têm como atividade montar o pacote com o número identificador e com a descrição. O tempo de montagem de cada pacote pode variar de 1 até 5 unidades de tempo. A função *mTempo()* nos modelos calcula o tempo de montagem de cada pacote. O modelo 1 faz a verificação de cada pacote, representado pela transição *Verificando Pacote*, e também leva de 1 a 5 unidades de tempo. Após isso, o modelo 1 despacha o pacote para uma fila para que ele seja enviado. O modelo 2 também monta pacote. Além disso, recebe os pacotes que estão na fila de espera e então envia dois pacotes por vez (transição *Enviando Pacotes*). O tempo de envio de pacotes leva de 3 a 7 unidades de tempo e é dado no modelo pela função *pTempo()*.

O lugar *Fila* e o lugar *Pacotes* são marcados com a *fusion tag "Fila de Espera"*, representada por um retângulo azul na figura 15. Quando dois ou mais lugares são marcados por uma mesma *fusion tag*, significa que aqueles lugares pertencem ao mesmo conjunto de fusão. Quando a transição *Despachar* for disparada, uma ficha é adicionada ao lugar *Fila* e então, automaticamente, uma ficha também aparecerá no lugar *Pacotes*,



(a) (b)

Figura 15 – Exemplo de um modelo usando a funcionalidade de fusion places disponível no CPN Tools.

como ilustra a figura 15(b). A implementação dos *color sets* e das funções usadas nos modelos da figura 15 são apresentados a seguir:

```
colset ID = INT;
colset DESC = with A | B | C | D;
colset PACOTE = product ID*DESC timed;
colset PAC_FINAL = product PACOTE*PACOTE timed;
colset TEMPO1 = int with 1..5;
colset TEMPO2 = int with 3..7;
var id:ID;
var desc:DESC;
var pac1,pac2: PACOTE;
fun mTempo() = TEMPO1.ran();
fun pTempo() = TEMPO2.ran();
```

Quando todos os membros de um conjunto de fusão pertencem a mesma parte ou página (no contexto de CPN Tools) da CP-net, e tem apenas uma instância, os *fusion places* são nada mais que um mecanismo conveniente para evitar vários cruzamentos de arcos (JENSEN; KRISTENSEN, 2009). Dessa forma, é possível simplificar a estrutura gráfica da rede sem mudar o seu significado. Para a utilização dos fusion places não é necessário nenhuma declaração especial. O software CPN Tools trás essa funcionalidade no menu de opções. O usuário deve então selecionar a opção desejada e aplicar a funcionalidade aos lugares que formarão o(s) conjunto(s) de fusão.

A aplicação prática da modelagem e análise de redes de Petri coloridas depende fortemente da existência de ferramentas computacionais que suportam a criação e manipulação dos modelos. O CPN Tools é uma ferramenta adequada para editar, simular e prover a

análise do espaço de estados (*state space*) de modelos CP-net (KRISTENSEN; CHRISTENSEN; JENSEN, 1998). O software possui interface gráfica eficiente, além de trazer a disposição dos usuários todas as funcionalidades necessárias para a edição de uma CP-net, desde paletas de criação, paletas de análise e simulação, e até paletas de configurações (estilo/preferências) para o modelo. A figura 16 ilustra a interface do Software CPN Tools bem como todas as opções de funcionalidades disponíveis para uso no menu da ferramenta.

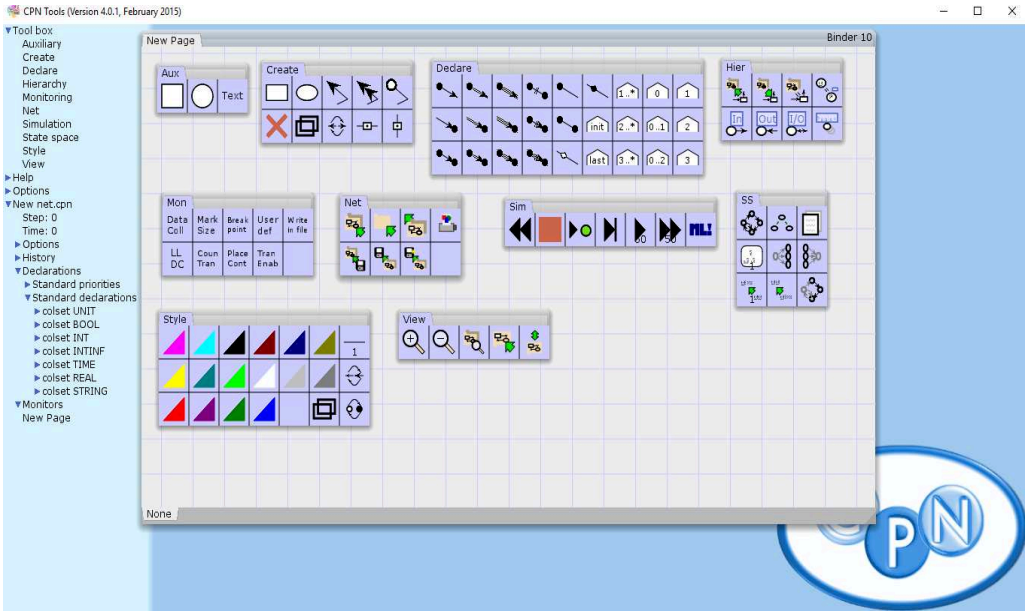


Figura 16 – Interface do Software CPN Tools.

2.6 Considerações Finais

Neste capítulo foram apresentados conceitos importantes para o desenvolvimento desta pesquisa. A começar pelo conceito clássico das redes de Petri. Foram apresentados a definição formal de uma rede de Petri, seus elementos básicos e o funcionamento de uma rede. Em seguida foram apresentadas algumas das propriedades principais das redes de Petri que servirão, nesta pesquisa, como parâmetros para análise dos modelos.

Este capítulo também trouxe o conceito de WorkFlow nets. As WF-nets são tipos particulares de redes de Petri desenvolvidas para modelar processos de negócios. Em especial, esse tipo de rede possui estruturas de roteamento específicas para representar a ordem em que as tarefas de um processo são executadas. Nesta pesquisa, as WF-nets serão utilizadas para modelar as diversas atividades de um jogo, bem como o fluxo das mesmas por meio das estruturas de roteamento. Além disso, a verificação da propriedade *soundness* da WF-net será realizada por meio de um algoritmo específico como método de análise do modelo criado para o jogo.

As WorkFlow nets Possibilísticas também foram apresentadas neste capítulo. Esse tipo de rede combina as estruturas de roteamentos das WF-net com o conceito de dispara impreciso das redes de Petri Possibilísticas. Assim, é possível lidar com modelos de sistemas complexos onde acontecimentos inesperados são recorrentes. As WF-nets Possibilísticas serão usadas nesta pesquisa para a representação dos modelos de jogos do tipo *multiplayer*. Esse tipo de jogo em particular permite a presença de vários jogadores ao mesmo tempo, interagindo uns com os outros e gerando interferências nas atividades de outros jogadores. Com o conceito de incerteza pertencente as WF-nets Possibilística, será então possível criar modelos de jogos *multiplayer* mais consistentes com a realidade.

O conceito de redes de Petri do tipo grafo de estados também foi apresentado. Em particular esse tipo de rede servirá para a representação do mundo virtual do jogo. As áreas do mapa, as fronteiras e a localização dos jogadores poderão ser representadas por esse tipo de rede de maneira simples e direta. O modelo gerado para o mundo virtual irá interagir diretamente com o modelo criado para representar as atividades do jogo. Dessa forma será possível propor um método de análise qualitativo e quantitativo considerando o comportamento global do jogo.

Por fim, os conceitos relacionados as redes de Petri Coloridas foram apresentados neste capítulo. Tais conceitos servirão para a implementação dos modelos gerados no software CPN Tools. Por meio da utilização do CPN Tools será possível não apenas criar os modelos como também utilizar funções específicas da ferramenta para simular, analisar e validar os modelos definidos nesta pesquisa.

Capítulo 3

Trabalhos Correlatos

No trabalho de Natkin, Vega e Grünvogel (2004), um novo método para auxiliar no processo de criação de *video games* é apresentado. Os autores abordam modelos para representar a lógica do jogo (sequência de ações) e o espaço virtual. Natkin, Vega e Grünvogel (2004) afirmam que modelos assim são importantes, pois a tarefa de criação de jogos consiste de estipular uma sequência de ações, em um ambiente simulado, que possua consistência e níveis de dificuldades satisfatórios para os usuários finais (jogadores). Para modelar a lógica do jogo, Natkin, Vega e Grünvogel (2004) utilizaram o formalismo das redes de Petri. Nessa abordagem, uma rede de Petri denominada Rede de Transações determina o início e o fim de cada ação do jogador. Assim, cada modelo representa um conjunto de atividades que podem ser executadas pelo jogador em determinado momento do jogo. Para modelar o espaço virtual, os autores utilizaram hipergrafos. Hipergrafos são generalizações de grafos. Seus nós (vértices) são conectados por hiperarestas que são definidas como um par ordenado de subconjuntos disjuntos de vértices. Uma hiperaresta pode conectar vários nós. Assim, cada vértice do hipergrafo representa uma região do jogo onde as ações são executadas, e as hiperarestas representam os caminhos entre essas regiões. Com o objetivo de unir as duas estruturas e representar o jogo num contexto global, Natkin, Vega e Grünvogel (2004) criaram um mecanismo denominado de *conexões*. Esse mecanismo trata de substituir uma hiperaresta do hipergrafo pela árvore de alcançabilidade de uma rede de Petri. A cada vez que uma tarefa é realizada, um lugar do mapa topológico é liberado e a hiperaresta substituída no modelo. A verificação do modelo do jogo é tratada de acordo com cada formalismo (a verificação acontece no modelo lógico ou no mapa topológico já que se tratam de formalismos distintos que não permitem a integração dos dois modelos numa visão única).

As redes de Petri para a modelagem de *video games* também foram utilizadas em Araújo e Roque (2009). Neste trabalho as redes de Petri foram exploradas para a modelagem de sistemas e fluxos de jogos. Por meio de um estudo de caso os autores apresentaram como as redes de Petri podem ser utilizadas com certas vantagens em relação a outras linguagens de modelagem. Araújo e Roque (2009) usaram diagramas baseados em

redes de Petri para representar as possíveis ações dos jogadores em relação à objetivos de jogo. Assim, cada opção de ação que um jogador tem é representada por um conjunto de lugares e transições na rede de Petri. Para cada opção existe um novo fluxo que produz atividades e novas opções. Os autores também apresentaram a modelagem baseada em uma abordagem hierárquica onde cada nível da hierarquia é composto por um conjunto específico de ações relacionadas. Araújo e Roque (2009) citam que usando os diagramas baseados em redes de Petri é possível simular o comportamento do jogo.

Já no trabalho de Oliveira, Julia e Passos (2011) é apresentada uma nova abordagem baseada em um tipo particular de rede de Petri, denominado WorkFlow net, para especificar cenários existentes em um jogo. Nesta abordagem, os autores utilizaram uma WorkFlow net para representar o fluxo de atividades que deve ser executado pelo jogador a fim de alcançar um objetivo específico no jogo. Esse fluxo de atividades é associado à noção de *quest*, ou seja, uma missão que o jogador deve executar. De acordo com Oliveira, Julia e Passos (2011), em termos de modelo, cada *quest* é um subprocesso de uma WorkFlow net maior. A integração de diversas *quests* formam então uma rede de *quests* e é por meio dela que o resultado global da análise do modelo do jogo é estabelecido. Como cada *quest* é um subprocesso, caso aconteça alguma mudança em uma *quest* já estudada, um novo estudo das boas propriedades será feito apenas para a *quest* que sofreu alteração. Nessa abordagem, Oliveira, Julia e Passos (2011) realizaram uma análise qualitativa por meio de árvores de prova da lógica linear. De acordo com os autores, a tradução dos modelos em redes de Petri para árvores da lógica linear tem o objetivo de provar a propriedade *soundness* da rede que corresponde à consistência do cenário modelado do ponto de vista do jogo.

A lógica linear também foi utilizada para análise de jogos por Lewis e Whitehead (2011). Os autores explicam que o interesse de usar a lógica linear para análise de um cenário é a possibilidade de expressar ações e recursos do jogo sem reter informações inúteis. Na lógica linear, os personagens e recursos são representados por átomos. Enquanto que as relações implicativas são usadas para projetar eventos. De acordo com Lewis e Whitehead (2011), esse método proporciona a verificação de propriedades de cenários de jogos que podem ser validadas antes da execução/implementação do jogo. As classes de propriedades principais citadas pelos autores são: jogabilidade (verifica se o curso do jogo está correto) e relevância do cenário (verifica se o jogo é interessante). Após expressar os cenários utilizando fragmentos da lógica linear, o modelo é traduzido numa rede de Petri. O modelo final obtido permite gerar então as possíveis narrativas para um dado cenário.

A pesquisa apresentada no trabalho de Lee e Cho (2012) aborda a criação de um mecanismo para gerar enredo de *quest* consistindo na representação de eventos baseados em redes de Petri. De acordo com os autores, um enredo de *quest* é uma sequência de eventos para alcançar um objetivo específico. E um evento consiste de ações que devem ser executadas. As rede de Petri são então utilizadas para representar um evento do jogo.

Na rede, um lugar representa uma pré-condição para uma ação ou o armazenamento do resultado de uma ação. Os arcos representam relacionamentos casuais entre uma ação e sua pré-condição. As transições representam as ações do jogador e as fichas representam o estado de uma ação em um evento. O gerador de enredo baseado em redes de Petri, proposto por Lee e Cho (2012), foi então experimentado em uma plataforma de jogo comercial. De acordo com os autores, além de ser aplicável, o método proposto proporciona um método baseado em roteiro mais formal. Segundo Lee e Cho (2012), as redes de Petri ajudaram a identificar elementos passivos (tais como as condições) e elementos ativos (tais como as ações) da história, facilitou a representação de eventos independentes, assim como a representação de restrições e a sincronização de eventos.

O estudo realizado por Barreto e Julia (2014) apresenta uma abordagem onde os cenários de *video game* são representados pela combinação de WorkFlow nets e por redes de Petri do tipo Grafo de Estados. A WorkFlow net é utilizada para representar as atividades que existem em um jogo. A noção de mapa que representa as áreas do mundo virtual onde o jogador pode executar as atividades é representada por um Grafo de Estados. Para especificar a comunicação entre os dois modelos, Barreto e Julia (2014) implementaram um mecanismo de comunicação síncrona baseado em redes de Petri Coloridas. Dessa forma foi possível obter modelos de níveis de jogo. Para verificar a corretude do modelo global obtido (modelo das atividades + modelo do mapa), Barreto e Julia (2014) usaram um algoritmo baseado na análise do espaço de estados (*state space*). Tal algoritmo pôde ser executado automaticamente por algumas funcionalidades existentes no CPN Tools.

O conceito de redes de Petri na modelagem de jogos também é usado por Reuter, Göbel e Steinmetz (2015) para apresentar uma abordagem que atua nos estágios iniciais do projeto de jogo, detectando erros estruturais em jogos *singleplayer* (um jogador) e *multiplayer* (vários jogadores). Para diferenciar os jogos *singleplayers* dos *multiplayers*, Reuter, Göbel e Steinmetz (2015) utilizaram as redes de Petri Coloridas que podem atribuir tipos as fichas e funções de guarda nas transições. Reuter, Göbel e Steinmetz (2015) determinam que um jogo pode ser dividido em cenas. Cada cena possui elementos como cômodos, jogadores e variáveis. Todos os elementos do jogo são então mapeados para construtores apropriados das redes de Petri. Por exemplo, os cômodos do jogo (e.g., um quarto ou um depósito) são representados pelos lugares da rede de Petri, assim como as ações (e.g., abrir um porta) e as variáveis que representam valores booleanos. Já os jogadores são representados por fichas e os eventos do jogo por transições. A criação da rede de Petri é adicionada como uma extensão da ferramenta StoryTec (REUTER; GÖBEL; STEINMETZ, 2015). Assim, qualquer jogo que for criado com essa ferramenta pode ser transformado automaticamente em uma rede de Petri Colorida. A rede gerada é então exportada para o formato XML (*Extensible Markup Language*) que pode ser lido pela ferramenta CPN Tools. De acordo com Reuter, Göbel e Steinmetz (2015), quando a análise do modelo é feita no CPN Tools, é possível identificar erros estruturais como

deadlocks (situações nas quais os jogadores não podem mudar o estado do jogo), *livelocks* (situações nas quais os jogadores podem mudar o estado do jogo, mas não podem alcançar o final), cenas inalcançáveis (situações nas quais os jogadores não podem chegar a uma cena sob quaisquer circunstâncias) e ações impossíveis (situações nas quais as ações nunca podem ser acionadas, devido a condições não satisfeitas).

Rezin et al. (2018), apresentaram uma abordagem para verificar formalmente jogos de computador do tipo *multiplayer*. O processo do jogo é representado por um autômato finito não determinístico que evolui a cada unidade de tempo com base nas ações dos jogadores ou em ações aleatórias integradas ao jogo. Para tanto, Rezin et al. (2018) definem formalmente um jogo *multiplayer* como um conjunto composto por atores (todos os objetos ativos que podem influenciar no processo do jogo), atributos (uma variável inteira que pode alterar o seu valor durante o jogo), parâmetros (uma constante designada antes do jogo começar e que não pode ser mudada, por exemplo o tamanho do mundo virtual ou a velocidade de alguns atores) e ações que podem alterar atributos e serem compartilhadas entre atores. Assim, o autômato representando o jogo é construído nesses termos. Devido a complexidade da construção de um autômato, Rezin et al. (2018) desenvolveram uma ferramenta que possa apoiar a construção e representação formal do modelo de jogo e depois verificar automaticamente as propriedades de interesse. Após a verificação, o modelo pode ser automaticamente traduzido para uma linguagem de programação de alto nível a fim de integrá-lo ao jogo em desenvolvimento. Dessa forma, o método proposto executa essencialmente os seguintes passos: (1) construir a definição formal do jogo a partir da descrição do jogo; (2) com a definição do jogo, construir uma estrutura específica (chamada de estrutura Kripke) para ser aplicada a uma ferramenta de verificação de modelo; (3) no caso da verificação falhar devido ao problema de explosão de estados o verificador de modelo irá fornecer um contra-exemplo para ser usado na fase de depuração, ou ainda técnicas de redução deverão ainda ser aplicadas ao modelo; (4) traduzir o modelo verificado para uma linguagem de programação específica e integrar o código em um projeto de jogo já existente.

3.1 Considerações Finais

Neste capítulo foram apresentados trabalhos que estão relacionados ao tema desta pesquisa e contribuem para o processo de desenvolvimento de jogos por meio de métodos formais. Em sua maioria, os trabalhos apresentam a utilização das redes de Petri como uma linguagem de modelagem eficiente para especificar e analisar formalmente *video games*.

A abordagem de modelagem apresentada por Natkin, Vega e Grünvogel (2004), por exemplo, é interessante a medida que define diagramas para representar aspectos do jogo que são importantes para o seu processo de criação, facilitando assim o estudo da joga-

bilidade. Um tipo de análise formal dos modelos pode então mostrar os problemas de jogabilidade ainda em nível de projeto. No entanto, o uso de dois formalismos diferentes (redes de Petri e hipergrafos) torna o estudo global do modelo de jogo inviável. Dessa forma, para verificar a corretude do jogo, é necessário analisar os modelos separadamente de acordo com seus formalismos.

Já o trabalho de Araújo e Roque (2009) apresenta uma construção que pode resultar em modelos bastante complexos a medida que vão crescendo. Ainda que sejam implementados em uma estrutura hierárquica, fica difícil visualizar o sistema inteiro. Apesar disso, os autores afirmaram que a notação gráfica das redes de Petri é simples e pode ser utilizada para modelar jogos complexos. Além disso, a estrutura matemática das redes de Petri permite que os sistemas modelados sejam formalmente analisados. Também a simulação dos comportamentos de jogo oferece a possibilidade de detectar problemas ainda na fase de projeto do jogo (ARAÚJO; ROQUE, 2009). Diferente de Natkin, Vega e Grünvogel (2004), Araújo e Roque (2009) apresentaram apenas a representação do fluxo de jogo.

A abordagem apresentada por Oliveira, Julia e Passos (2011) utiliza as WorkFlow nets para representar as *quests* de um jogo. No entanto, os autores não apresentaram um modelo para representar o mapa do mundo virtual do jogo onde as ações do jogador são executadas. Além disso, os autores apresentaram uma análise qualitativa usando um formalismo diferente das redes de Petri, sendo necessário assim realizar a tradução da rede de Petri para o formalismo da lógica linear. Em contrapartida, a abordagem proposta por Barreto e Julia (2014) apresentou a modelagem das ações por meio do modelo de atividades, e o mapa do mundo virtual do jogo por meio do modelo topológico. Ainda em Barreto e Julia (2014), a análise dos modelos é executada dentro de um só formalismo e com o auxílio de ferramentas computacionais baseadas em redes de Petri Coloridas. Entretanto, essa abordagem trata apenas de jogos *singleplayers* e não aborda outros aspectos do jogo (como noção de itens e tempo, por exemplo).

Reuter, Göbel e Steinmetz (2015) afirmam em seu trabalho que o modelo de verificação em rede de Petri gerado automaticamente a partir do jogo não possui margem para inconsistência entre o jogo e o modelo, o que pode acontecer quando o modelo é criado em paralelo pelos próprios desenvolvedores. Apesar da geração automática do modelo, essa abordagem apresenta um modelo complexo. Por exemplo, quando é necessário representar uma situação de escolha entre diferentes opções no jogo, o modelo duplica a quantidade de estados de acordo com os valores que a variável consegue assumir. Assim, o modelo gerado precisa passar por estratégias de otimização para ser analisado formalmente (REUTER; GÖBEL; STEINMETZ, 2015).

O mesmo problema acontece em Rezin et al. (2018). O método apresentado por Rezin et al. (2018) apesar de fazer uma verificação formal tem que lidar constantemente com o problema da explosão de estados pelo autômato finito não-determinístico. Além disso, o

processo de verificação apresentado pelos autores não é rápido e precisa de aprimoramentos para fornecer um procedimento de testes mais confiável (REZIN et al., 2018).

Considerando a literatura correlata aqui apresentada, esta pesquisa propõe um método para a verificação e análise dos modelos dos cenários de jogos utilizando para isso, além do formalismo das redes de Petri, ferramentas computacionais baseadas em redes de Petri Coloridas. O método proposto também apresenta a definição de um modelo temporizado de cenários de vídeo games para estimar a duração de um jogo. Além disso, o método proposto neste trabalho apresenta um modelo para formalizar cenários de jogos *multiplayers*. Todos os modelos definidos por esta abordagem utilizam, além dos fundamentos da Teoria das redes de Petri, ferramentas existentes prontas de modelagem, análise e simulação baseadas em redes de Petri como, por exemplo, PIPE2 e CPN Tools. Além de possuírem funcionalidades necessárias para a manipulação de modelos baseados em rede de Petri, essas ferramentas são softwares livres.

Capítulo 4

Modelagem de Video Games baseado em Redes de Petri no caso Singleplayer

De modo geral, pode-se dizer que vídeo games são um tipo especial de aplicação multimídia de entretenimento que requer participação ativa do usuário (CALLELE; NEUFELD; SCHNEIDER, 2005). É importante garantir que a experiência do jogo leve a uma sucessão de metas dentro de um prazo razoável. O jogador precisa ter a sensação de liberdade no mundo virtual, mesmo quando ele é orientado para uma solução de uma forma inconsciente (ROLLINGS; MORRIS, 2003). Cumprir todas as exigências para fazer com que um jogo seja atrativo não é uma tarefa trivial. Dessa forma, os projetos de jogos são tarefas complexas. Portanto, é necessário encontrar novos conceitos e ferramentas para suprir os desafios do desenvolvimento de jogos, como afirmam Reyno e Cubel (2009), Kanode Christopher M. e Haddad (2009) e Lewis e Whitehead (2011).

Este capítulo tem como ponto de partida a abordagem proposta por Barreto e Julia (2014) para a modelagem de cenários de vídeo games usando o formalismo das redes de Petri. Um jogo é considerado como uma coleção de níveis onde cada nível possui atividades que precisam ser executadas pelo jogador. Essas atividades são representadas nesta abordagem por um tipo particular de rede de Petri denominada WorkFlow net. A execução das atividades de um jogo depende também de um mundo virtual representado por um mapa topológico. Tal mapa será ilustrado nessa abordagem por um modelo do tipo grafo de estados. Dessa forma, cada nível do jogo será apresentado pela colaboração entre um modelo lógico (atividades) e um modelo topológico (mapa do mundo virtual). A junção desses dois modelos será apresentada nessa abordagem por meio de um mecanismo de comunicação assíncrono, também baseado nas redes de Petri.

Para a ilustrar o abordagem, será utilizado o jogo Silent Hill II. Silent Hill é um jogo de aventura/terror que começa em uma cidade deserta. O jogador assume o papel de James, um personagem que não possui objetivos muito claros no começo do jogo. James recebe uma estranha carta de sua esposa chamando-o para se encontrarem em Silent Hill. No entanto, a esposa de James está morta a três anos. Incentivado pelo mistério da carta,

James retorna a Silent Hill. Para alcançar o objetivo do jogo, o jogador deve explorar o ambiente, lutar contra inimigos, coletar objetos e resolver enigmas.

4.1 Modelo de Análise

4.1.1 Modelo Lógico

Para a representação do modelo lógico, é necessário considerar o conceito de nível de jogo. Um nível de jogo representa uma parte ou estágio do jogo. Os jogos, em sua maioria, podem ser estruturados em grupos de níveis. Cada nível possui um objetivo associado, uma sequência de atividades e uma topologia (mundo virtual). O objetivo se trata da missão que o jogador precisa executar para vencer. Para alcançar o objetivo é necessário que o jogador execute uma sequência de atividades. No trabalho de Barreto e Julia (2014), uma atividade é considerada como um elemento atômico da história ou alguma coisa significante que acontece. Dessa forma, é importante considerar a lógica do sequenciamento das atividades em um nível de jogo.

No presente trabalho, o termo *atividade* será adotado para descrever um evento atômico do jogo (geralmente alguma ação que o jogador precisa executar). O termo *missão*, ou *quest*, será usado para representar um objetivo de jogo que precisa ser alcançando após a execução de um conjunto específico de atividades. Já o termo nível será usado para expressar as partes (ou estágio) do jogo. Geralmente um nível de jogo contém pelo menos uma missão, e uma missão contém pelo menos uma atividade associada.

Uma sequência de atividades, no contexto de um nível de jogo, possui um início e um fim bem definidos. As atividades podem ser obrigatórias ou opcionais. A execução das atividades pode ser realizada sequencialmente ou em paralelo. Assim, é possível definir as propriedades esperadas de um nível de jogo da seguinte forma:

- ❏ existe um momento específico que determina o início das atividades do nível. Da mesma forma, existe um momento específico que determina o final do nível;
- ❏ todas as atividades do nível precisam estar entre o início e o fim;
- ❏ as atividades de um nível podem ser divididas entre atividades obrigatórias e atividades opcionais.

É possível associar a estrutura das atividades de um jogo com a estrutura clássica de um processo de negócio, pois ambos possuem um começo e um objetivo final que será alcançado após a execução de várias atividades. Nesse sentido, as WorkFlow nets são adequadas para a modelagem de níveis de vídeo games.

A modelagem de uma sequência de atividades de um jogo em termos de uma WorkFlow net é dada da seguinte forma: as atividades são modeladas por transições, os lugares

representam as condições para iniciar as atividades e os jogadores são modelados por fichas. O modelo lógico do jogo pode ser definido como se segue.

Definição 10 (Modelo Lógico) *Um Modelo Lógico de um nível de jogo é uma Work-Flow net ML = (P,T,J), tal que:*

- ❑ ***P*** *é um conjunto finito de lugares, onde:*
  - *existe um único lugar de início $i \in P$ tal que $\bullet i = \emptyset$;*
  - *existe um único lugar de fim $o \in P$ tal que $o\bullet = \emptyset$;*
  - *todo nó $x \in P \cup T$ está em um caminho entre os lugares i e o.*
- ❑ ***T*** *é o conjunto finito de transições da rede que representam as atividades;*
- ❑ ***J*** *é o conjunto de marcações que representa o jogador.*

Para modelar um nível de jogo, é necessário identificar as atividades do nível. Considere o primeiro nível do jogo Silent Hill II como exemplo. O modelo lógico do primeiro nível de Silent Hill II é apresentado na figura 17. As atividades que o jogador precisa executar para terminar o primeiro nível do jogo são:

1. encontrar o mapa de Silent Hill na plataforma de observação;
2. matar a criatura;
3. pegar o rádio no túnel;
4. encontrar o bilhete no trailer;
5. encontrar um mapa no bar;
6. encontrar a chave na rua Martin;
7. entrar no apartamento Wood Side.

A figura 17 ilustra uma WorkFlow net que representa o fluxo de atividades do nível. O conjunto de lugares da rede é dado por P = {*P0*, *P1*, *P2*, *P3*, *P4*, *P5*, *P6*, *P7*, *P8*, *P9*, *P10*}. O conjunto de transições é dado por T = {*Encontrar mapa de Silent Hill*, *Matar criatura*, *Pegar o rádio*, *Encontrar bilhete*, *Encontrar mapa no bar*, *Encontrar chave*, *Ir ao apartamento*, *Fork*, *Join*}. O momento de início do nível é representado pelo lugar *P0* e o fim do nível é representado pelo lugar *P10*. Portanto, a ficha no lugar *P0* representa o jogador no início do nível.

No primeiro nível de Silent Hill II todas as atividades são obrigatórias. Todas as atividades são sequenciais, exceto as atividades *Matar criatura* e *Pegar o rádio* que podem

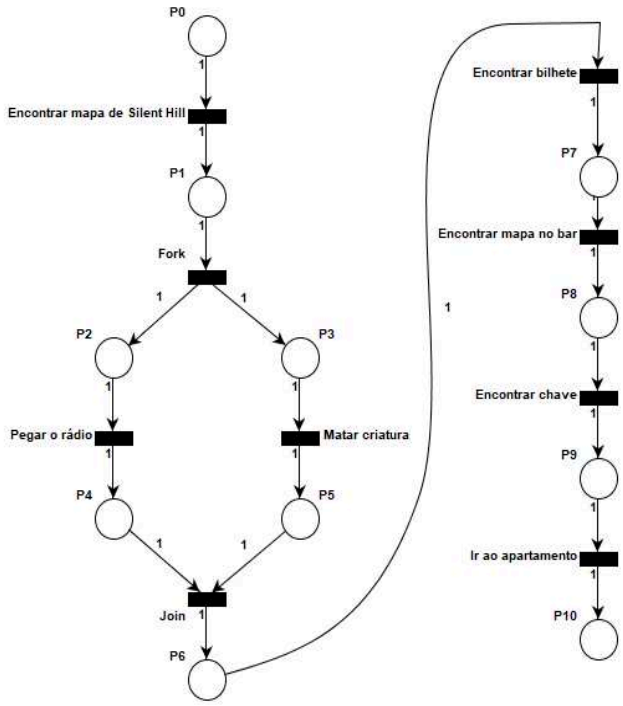


Figura 17 – Modelo lógico do primeiro nível do jogo Silent Hill II.

ser realizadas em qualquer ordem. Na figura 17 a transição *Fork* representa o início do comportamento paralelo dessas duas atividades, enquanto que a transição *Join* representa a conclusão dessas atividades.

Barreto e Julia (2014) apresentaram uma abordagem interessante usando o software CPN Tools para a modelagem do Modelo Lógico. Nas redes de Petri ordinárias, as fichas são indistinguíveis. Já em uma rede de Petri Colorida, as fichas podem ser associadas a tipos de dados (cores). Assim, cada lugar de um modelo Colorido precisa ter um tipo de dado associado a ele, denominado de *color set*. Portanto, para adaptar um modelo lógico para a linguagem do CPN Tools é preciso definir um tipo de dado para as fichas e os lugares da rede. Também é necessário definir uma variável do mesmo tipo para associá-la aos arcos do modelo Colorido. Portanto, a definição do Modelo Lógico adaptado para o CPN Tools é apresentada a seguir.

O modelo apresentado na figura 18 ilustra a versão do Modelo Lógico para o CPN Tools do primeiro nível de Silent Hill II. O modelo da figura 18 é equivalente ao modelo da figura 17. Assim, o conjunto de lugares e o conjunto de transições de ambas as redes permanecem os mesmos, ou seja, P = {*P0*, *P1*, *P2*, *P3*, *P4*, *P5*, *P6*, *P7*, *P8*, *P9*, *P10*} e T = {*Encontrar mapa de Silent Hill*, *Matar criatura*, *Pegar o rádio*, *Encontrar bilhete, Encontrar mapa no bar*, *Encontrar chave*, *Ir ao apartamento*, *Fork*, *Join*}, respectivamente. O conjunto de cores é dado por Σ = {PLAYER}. O *color set* PLAYER possui valor do tipo *player* e é associado a todos os lugares da rede. O conjunto de fichas é dada por J = {1‘player}

e o conjunto de variável é dado por V = {p}, sendo Tipo[p] = PLAYER. A variável p é associada a todos os arcos do modelo.

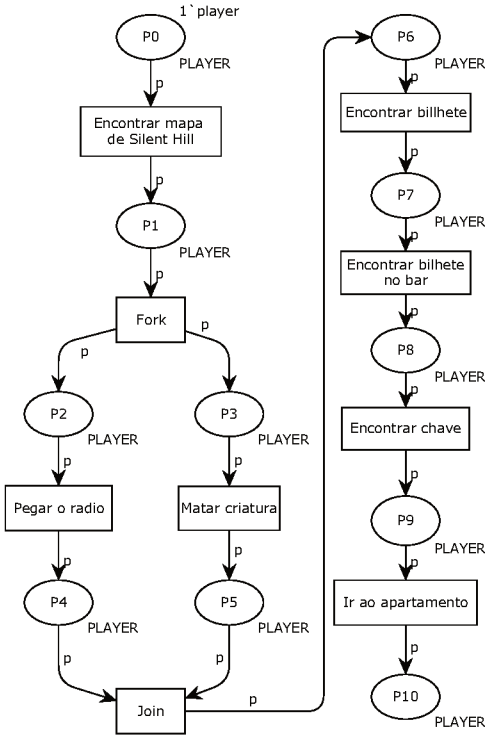


Figura 18 – Rede de Petri Colorida do Modelo Lógico do primeiro nível do jogo Silent Hill II.

4.1.2 Modelo Topológico

A topologia de um jogo refere-se ao mundo virtual criado para a imersão do jogador dentro do universo do jogo. O mundo virtual é composto por diferentes áreas. Cada área pode ser um lugar específico como um quarto, por exemplo, ou ainda um conjunto de lugares que o jogador poderá acessar como, por exemplo, o interior de uma casa ou o centro de uma cidade. Cada atividade ou missão do jogo é executada pelo jogador em uma área específica do mapa. Assim, a descrição das propriedades do mundo virtual bem como a evolução do jogador dentro deste mundo são indispensáveis para o processo de criação do jogo.

O mundo virtual será descrito como um mapa topológico que representa ambientes do mundo virtual como uma lista de áreas significativas para o jogo. As áreas são conectadas entre si. O jogador poderá ir de uma área para outra livremente ou satisfazendo algumas condições específicas, dependendo do objetivo do jogo. Assim, é possível descrever as características de um mapa topológico de jogo da seguinte forma:

- as áreas de um mapa são fixas durante todo o jogo. Isso quer dizer que o mundo virtual de um jogo é o mesmo do começo ao fim;
- existem áreas no mapa que são significativas, ou seja, áreas onde acontecem as *quests*;
- existem ligações entre áreas adjacentes;
- as áreas do mapa podem ser de livre acesso ou não. No caso de uma área bloqueada, é preciso que o jogador cumpra alguma condição específica para desbloquear a área em questão.

É possível representar o mapa topológico de um jogo utilizando uma rede de Petri do tipo grafo de estado. Em um grafo de estado, uma área significativa do mapa topológico é modelada por um lugar específico. As fronteiras entre áreas adjacentes são representadas por transições simples. Cada transição da rede que representa o mapa topológico tem apenas um lugar de entrada e um lugar de saída. A localização do jogador é representada por uma ficha em um lugar específico da rede; já a orientação do arco representa em qual direção o jogador pode ir entre duas áreas adjacentes. Sendo assim, o modelo topológico pode ser definido como se segue.

Definição 11 (Modelo Topológico) *Um Modelo Topológico (MT) de um jogo é uma rede de Petri do tipo grafo de estado fomada pela tupla (P, T, I, O, C), tal que:*

- ***P** é o conjunto de áreas significativas do mapa de jogo;*
- ***T** é o conjunto de fronteiras entre áreas adjacentes do mapa de jogo;*
- ***I** é a relação que define para cada fronteira t_j, uma única área de entrada para a fronteira $I(t_j)$;*
- ***O** é a relação que define para cada fronteira t_j, uma única área de saída para a fronteira $O(t_j)$.*

Um mapa topológico de jogo pode ser dividido em partes de acordo com cada nível. Para tanto, inicialmente, é preciso definir o conjunto de áreas significativas. As áreas significativas são lugares no mapa onde o jogador irá executar as atividades do jogo. Considere o exemplo do primeiro nível do jogo Silent Hill II. A figura 19 ilustra uma parte do mapa do jogo.

Os número circulados na figura representam as áreas significativas do mapa para o primeiro nível do jogo Silent Hill II, são elas:

1. plataforma de observação;
2. floresta;

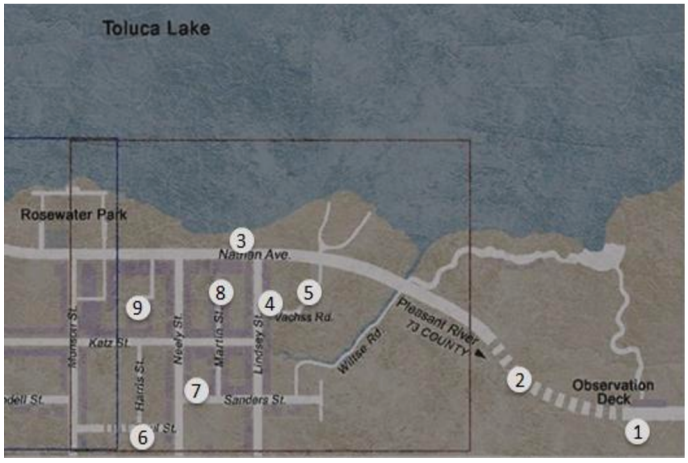


Figura 19 – Mapa parcial do primeiro nível de Silent Hill II.

3. igreja;
4. quintal;
5. túnel;
6. trailer;
7. bar;
8. rua Martin;
9. apartamento.

A figura 20 ilustra o grafo de estado que representa o mapa topológico para o primeiro nível de Silent Hill II. O conjunto de áreas significativas do mapa é dado por P = {Plataforma de observação, Floresta, Igreja, Túnel, Trailer, Bar, Quintal, Rua Martin, Apartamento}, e é representado pelos lugares da rede. O conjunto de fronteiras entre as áreas adjacentes é dado por T = {*T0, T1, T2, T3, T4, T5, T6, ..., T30*}, e é representado pelas transições. A relação I e a relação O que definem a área de entrada e saída, respectivamente, para cada fronteira da figura 20 é dada por:

- I(T0) = {Plataforma de observação}, O(T0) = {Floresta};
- I(T1) = {Floresta}, O(T1) = {Plataforma de observação};
- I(T2) = {Floresta}, O(T2) = {Igreja};
- I(T3) = {Igreja}, O(T3) = {Floresta};
- I(T4) = {Igreja}, O(T4) = {Quintal};
- I(T5) = {Quintal}, O(T5) = {Igreja};

- ❏ I(T6) = {Quintal}, O(T6) = {Túnel};
- ❏ I(T7) = {Túnel}, O(T7) = {Quintal};
- ❏ I(T8) = {Trailer}, O(T8) = {Bar};
- ❏ I(T9) = {Bar}, O(T9) = {Quintal};
- ❏ I(T10) = {Quintal}, O(T10) = {Rua Martin};
- ❏ I(T11) = {Rua Martin}, O(T11) = {Apartamento};
- ❏ I(T12) = {Bar}, O(T12) = {Trailer};
- ❏ I(T13) = {Quintal}, O(T13) = {Bar};
- ❏ I(T14) = {Rua Martin}, O(T14) = {Quintal};
- ❏ I(T15) = {Trailer}, O(T15) = {Igreja};
- ❏ I(T16) = {Igreja}, O(T16) = {Trailer};
- ❏ I(T17) = {Bar}, O(T17) = {Igreja};
- ❏ I(T18) = {Igreja}, O(T18) = {Bar};
- ❏ I(T19) = {Rua Martin}, O(T19) = {Igreja};
- ❏ I(T20) = {Igreja}, O(T20) = {Rua Martin};
- ❏ I(T21) = {Quintal}, O(T21) = {Trailer};
- ❏ I(T22) = {Trailer}, O(T22) = {Quintal};
- ❏ I(T23) = {Trailer}, O(T23) = {Rua Martin};
- ❏ I(T24) = {Rua Martin}, O(T24) = {Trailer};
- ❏ I(T25) = {Trailer}, O(T25) = {Apartamento};
- ❏ I(T26) = {Bar}, O(T26) = {Rua Martin};
- ❏ I(T27) = {Rua Martin}, O(T27) = {Bar};
- ❏ I(T28) = {Bar}, O(T28) = {Apartamento};
- ❏ I(T29) = {Quintal}, O(T29) = {Apartamento};
- ❏ I(T30) = {Igreja}, O(T30) = {Apartamento}.

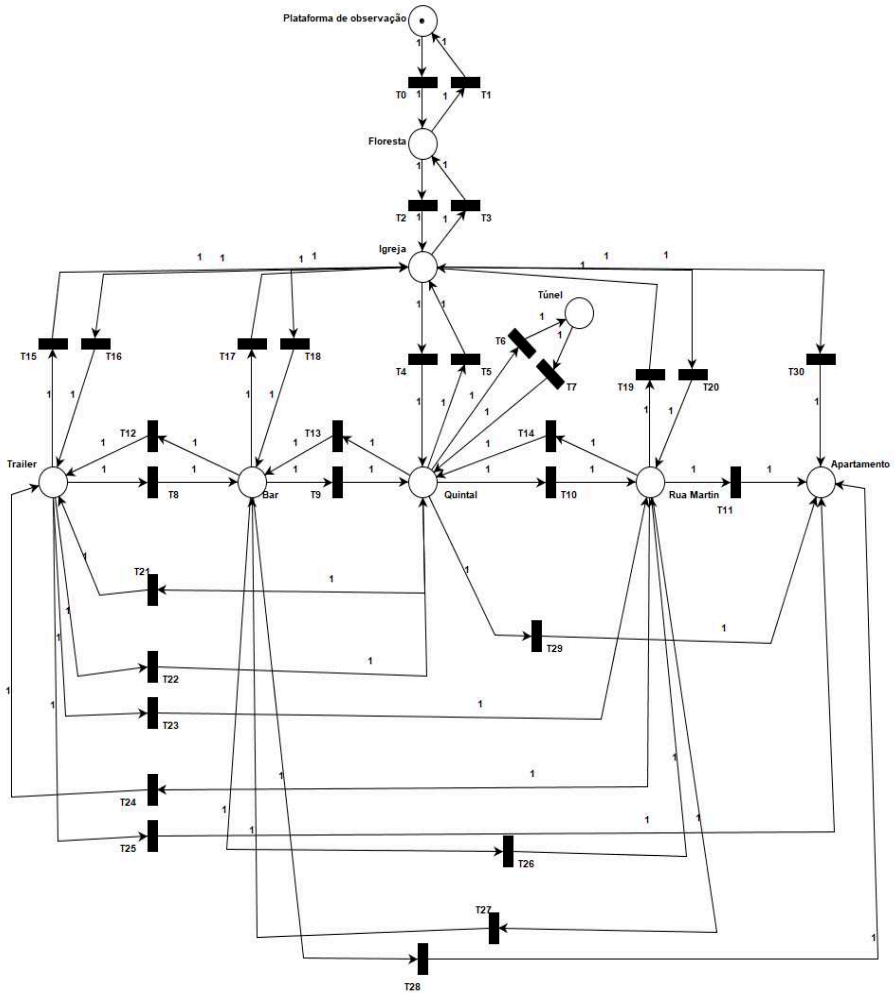


Figura 20 – Grafo de estado do mapa topológico de Silent Hill II.

Assim é possível ver quais áreas são adjacentes. Por exemplo, a área *Igreja* é adjacente as áreas *Floresta*, *Quintal*, *Bar*, *Trailer*, *Rua Martin* e *Apartamento*. Inicialmente, o jogador está localizado na área *Plataforma de observação*. Uma ficha nesse lugar representa a localização atual do jogador no início do jogo. De acordo com a execução das atividades, o jogador mudará sua localização. Entretanto, no começo, o jogador não pode acessar todas as áreas conectadas. Para passar de uma área para outra é preciso respeitar algumas condições (requisitos) que dependerão de atividades associadas ao modelo lógico do nível.

É possível também adaptar o Modelo Topológico para o CPN Tools. Neste caso o modelo necessitará de um tipo de dado para os lugares da rede e para as fichas. Também é necessário definir uma variável que será associada aos arcos do modelo Colorido.

A figura 21 ilustra a versão em rede de Petri Colorida do Modelo Topológico do jogo. O modelo da figura 21 é equivalente ao modelo apresentado na figura 20. O conjunto de áreas significativas P, o conjunto de fronteiras T e as relações I e O são os mesmos

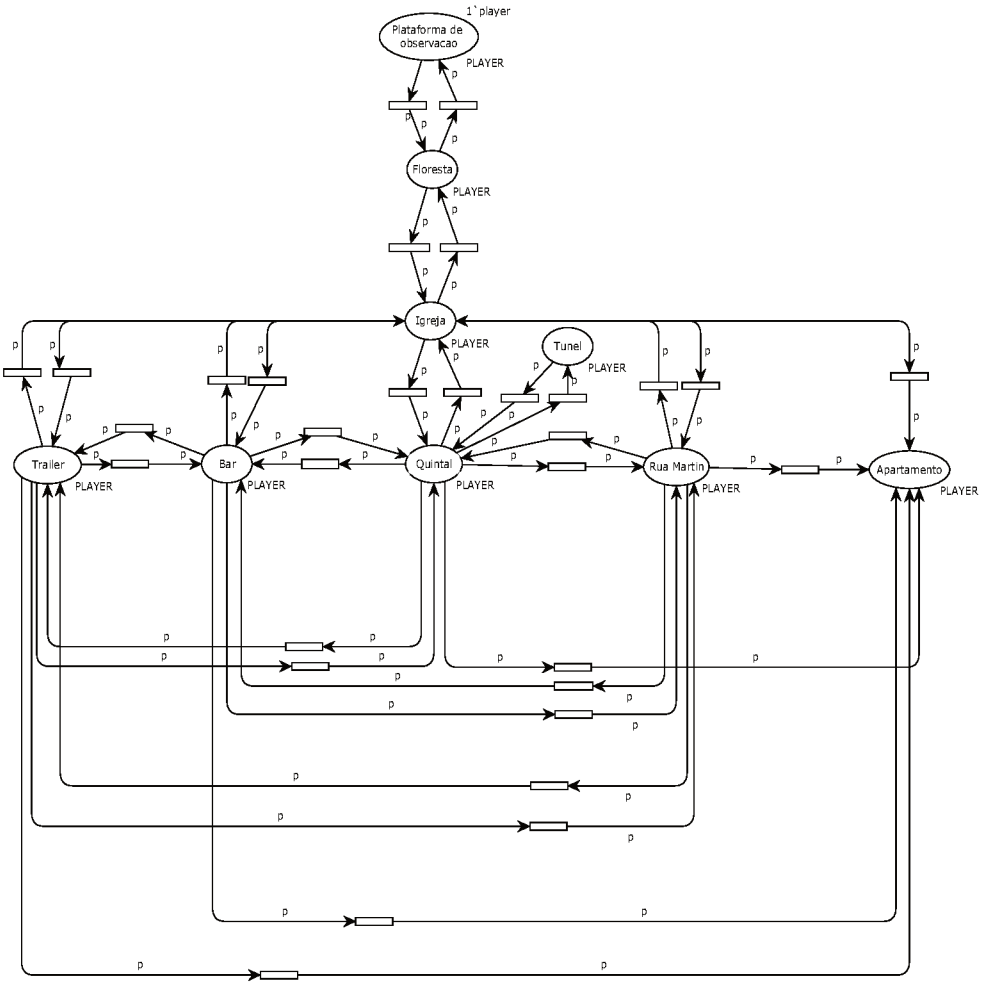


Figura 21 – Rede de Petri Colorida do Modelo Topológico do primeiro nível do jogo Silent Hill II.

apresentados para a figura 20. O conjunto de cores é dado por $\Sigma = \{\text{PLAYER}\}$. O conjunto de variáveis é dado por V = {p} onde Tipo[p] = PLAYER. A variável p é associada aos arcos da rede.

4.1.3 Mecanismo de Comunicação

A maior parte das atividades de um jogo só pode ser executada se o jogador estiver em um lugar específico do mundo virtual. Por outro lado, grande parte das áreas do mundo virtual só poderão ser acessadas após a realização de algumas atividades específicas do jogo. Dessa forma, é possível identificar a relação existente entre o Modelo Lógico e o Modelo Topológico do jogo.

Em sistemas de engenharia é comum trabalhar com modelos que se comunicam por meio de mensagens síncronas e assíncronas (ANDREU; PASCAL; VALETTE, 1996). Dado uma WorkFlow net que define o Modelo Lógico de um nível de jogo e um grafo de estado que representa o mapa topológico correspondente, é possível aplicar o mesmo princípio para a modelagem de vídeo games (BARRETO; JULIA, 2014). Uma das van-

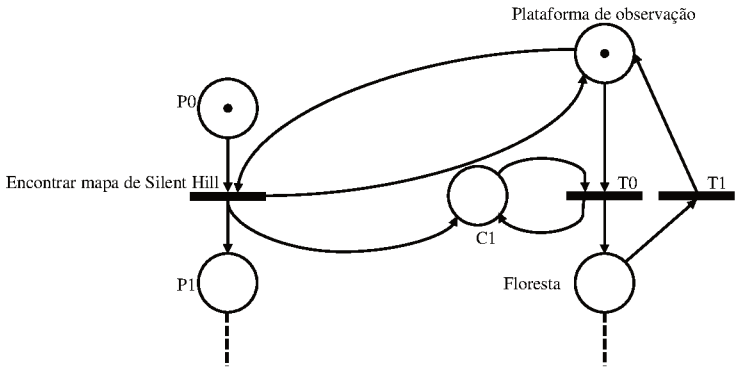
tagens em trabalhar com modelos distintos que se comunicam é que cada modelo pode ser redefinido sem alterar outro modelo existente.

Considere como exemplo o início do jogo Silent Hill II, onde o jogador se encontra na área *Plataforma de observação*. Inicialmente, essa é a única área que o jogador pode ter acesso. A condição para o jogador deixar a primeira área é encontrar o mapa de Silent Hill (primeira atividade do jogo). E para encontrar o mapa de Silent Hill, o jogador precisa estar localizado na área *Plataforma de observação*. Uma forma de representar formalmente esse tipo de condição, pode ser por meio do disparo de uma transição específica do Modelo Lógico que representará a produção do recurso correspondente. É possível estabelecer então um mecanismo de comunicação assíncrona entre o Modelo Lógico e o Modelo Topológico. A figura 22 ilustra um modo formal para especificar tal mecanismo, considerando o exemplo descrito acima.

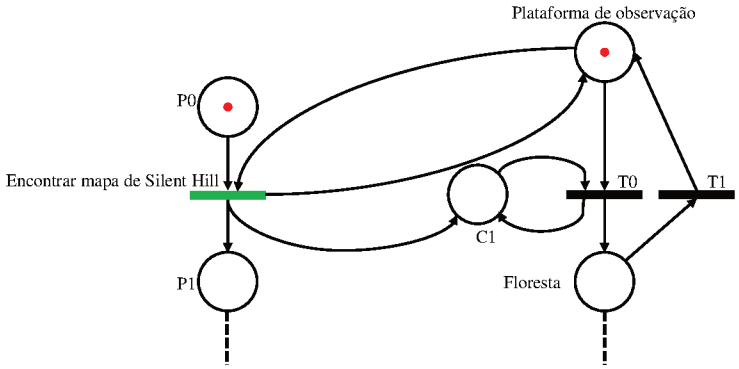
No lado esquerdo da figura 22(a) está representado um fragmento do Modelo Lógico. No lado direito da figura está representado parte do Modelo Topológico. Neste caso, a transição *Encontrar mapa de Silent Hill* é habilitada se existir uma ficha no lugar *P0* e uma ficha no lugar *Plataforma de observação* (como ilustra a figura 22(b)). Depois de disparar a transição, são produzidas três fichas. Uma ficha retorna para o lugar *Plataforma de observação*, outra é produzida em *P1* (continuação do Modelo Lógico) e outra em *C1* (condição para a liberação da área *Floresta*), como pode ser visto na figura 22(c).

Para qualquer cenário de jogo, onde uma área do mapa topológico só poderá ser desbloqueada mediante a realização de alguma atividade que satisfaz uma condição, é possível utilizar o mecanismo de comunicação assíncrona ilustrado na figura 22 e generalizado na figura 23. Na figura 23, a transição *Tx* representa alguma atividade do Modelo Lógico. Já a transição *Ty* representa uma passagem entre duas áreas adjacentes no Modelo Topológico. O lugar *Cx* representa uma condição necessária para autorizar a passagem entre áreas representada por *Ty*. Na maioria dos casos, essas condições corresponderão a um objeto necessário para passar de uma área para outra, como uma chave por exemplo. Por outro lado, algumas condições corresponderão simplesmente à realização de uma atividade que o jogador deverá executar e não necessariamente à produção de algum item específico do jogo. Portanto, na figura 23, o disparo da transição *Tx* representará a produção do recurso, já o lugar *Cx* marcado representará uma condição válida necessária para habilitar a transição *Ty*. Uma vez que uma condição é satisfeita, a passagem correspondente entre áreas adjacentes é liberada, e continuará até que o nível seja completado. Isso significa que quando uma condição é satisfeita (uma ficha produzida no lugar da condição), o lugar correspondente à condição continuará marcado o tempo todo.

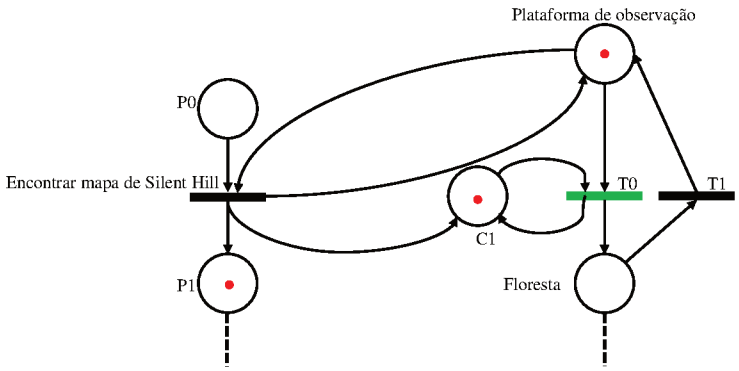
No caso de uma atividade de um jogo qualquer que só pode ser executada quando o jogador se encontra numa área específica daquele jogo, o mecanismo de comunicação poderá ser estabelecido pelo esquema de construção ilustrado na figura 24. Uma condição C poderá ser ou não obrigatória, dependendo do jogo em questão. Se a execução de uma



(a)



(b)



(c)

Figura 22 – Comunicação entre o Modelo Lógico e o Modelo Topológico.

atividade não é condição necessária para liberar uma fronteira entre áreas adjacentes, então tal atividade não fará parte do mecanismo de comunicação.

Também é possível adaptar o mecanismo de comunicação para o CPN Tools. É interessante utilizar recursos gráficos disponíveis na ferramenta para tornar a comunicação entre os modelos (lógico e topológico) visualmente independente. Portanto, Barreto e Julia (2014) utilizaram o conceito de *fusion places* (lugares de fusão), proposto pela ferramenta, tornando os dois modelos distintos (pelo menos visualmente).

Para a implementação do Mecanismo de Comunicação no CPN Tools, é necessário definir um tipo de dado específico para os lugares que representarão um requisito para a

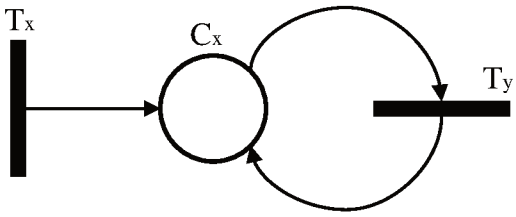


Figura 23 – Mecanismo de comunicação assíncrona para um cenário de vídeo game.

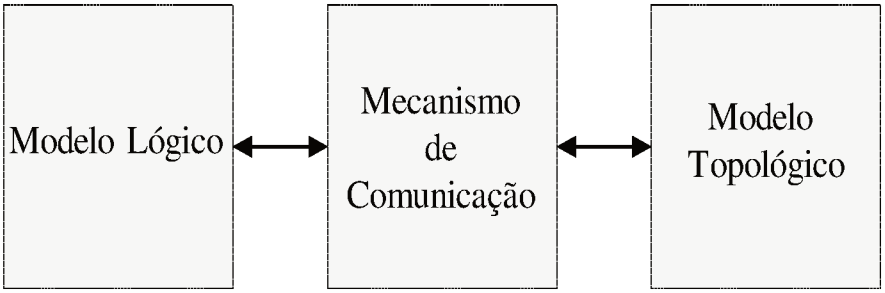


Figura 24 – Esquema para a construção de um cenário de vídeo game.

liberação de uma área do mapa. Também é necessário definir uma variável do mesmo tipo que será associada a todos os arcos de entrada e saída desses lugares. Portanto, o *color set COND* é definido e possui valores do tipo *condicao*[1]. Esse *color set* será associado a todos os lugares da rede que indicam a realização de uma atividade que implicará na liberação de uma área do mapa topológico. A variável *c* é definida e pertence ao Tipo[c] = COND.

A figura 25 ilustra o mecanismo de comunicação implementado no CPN Tools. Tal mecanismo é equivalente ao apresentado na figura 22. Os lugares *Plataforma de observacao*, *plataforma de observacao*, *C1* e *c1* são definidos como *fusion places* e marcados com um rótulo, denominado *fusion tag*. Graficamente, uma *fusion tag* é representada por um retângulo azul associado ao lugar. Lugares marcados com a mesma *fusion tag* pertencem ao mesmo conjunto de fusão *F*. Isso significa que se uma ação é executada em um lugar específico, todos os outros lugares da rede marcados com a mesma *fusion tag* também receberão a mesma ação.

Considere o exemplo da figura 25, após o disparo da transição *Encontrar mapa de Silent Hill*; uma ficha é produzida no lugar *c1* do Modelo Lógico. Uma mesma ficha é então reproduzida automaticamente no lugar correspondente *C1* do Modelo Topológico. Os lugares *c1* e *C1* estão marcados com a *fusion tag Mapa encontrado*, o que significa que esses lugares pertencem aos mesmo conjunto de fusão. Dessa forma, o mecanismo de comunicação é dado por um conjunto formado por transição e lugar.

[1] A ferramenta CPN Tools não aceita a inserção de caracteres especiais como os acentos, por exemplo.

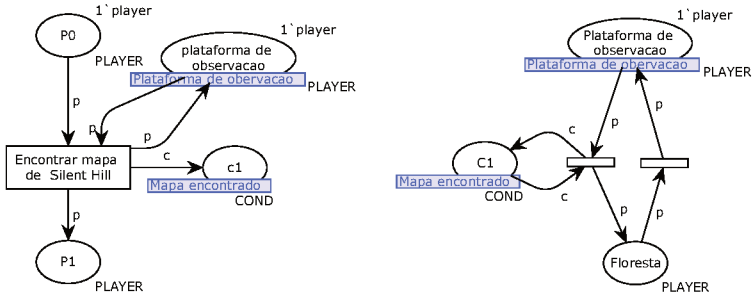


Figura 25 – Rede de Petri Colorida do mecanismo de comunicação assíncrona do primeiro nível do jogo Silent Hill II.

A figura 26 representa a comunicação completa entre o Modelo Lógico e o Modelo Topológico usando o conceito de *fusion place*. Com a utilização de *fusion place*, os dois modelos que se comunicam são graficamente independentes. O CPN Tools possibilita implementar cada modelo separadamente e permitir modificá-los quando necessário, como acontece no caso dos modelos que usam formas de comunicações assíncronas. Portanto, o uso de *fusion places* na abordagem apresentada por Barreto e Julia (2014) age como uma conveniência gráfica que permite evitar vários cruzamentos de arcos, simplificando assim a estrutura gráfica da rede sem alterar o seu significado.

A vantagem de se utilizar dois modelos distintos é que o modelo lógico pode ser alterado sem que o modelo topológico também o seja. O mesmo vale para o modelo topológico; é possível redefini-lo sem que o modelo lógico sofra alterações. Uma vez que existe modelos distintos para expressar as atividades e para expressar o mapa do mundo virtual, é possível simular então o funcionamento do jogo. Assim, o papel do mecanismo de comunicação é fazer com que os dois modelos trabalhem juntos, a fim de verificar o comportamento do jogo. É por meio da verificação do comportamento do jogo que os problemas podem ser encontrados e corrigidos ainda na fase de projeto ou antes da criação de um protótipo (BARRETO; JULIA, 2014).

4.1.4 Análise Qualitativa

A partir de um Modelo Global do jogo (Modelo Lógico + Modelo Topológico + mecanismo de comunicação) é possível realizar um tipo de análise qualitativa do jogo levando em consideração o ponto de vista de algumas propriedades importantes que garantem o bom funcionamento do jogo. Assim, o objetivo de se criar um modelo para análise de um vídeo game é garantir a consistência das atividades do jogo, bem como o acesso às áreas do mundo topológico.

O modelo para análise do jogo proposto por Barreto e Julia (2014) é baseado no método de verificação da propriedade *soundness*, definido por Aalst (1997). O Modelo de Análise propõe criar um Modelo Global estendido do jogo $\overline{MG}$, obtido adicionando uma

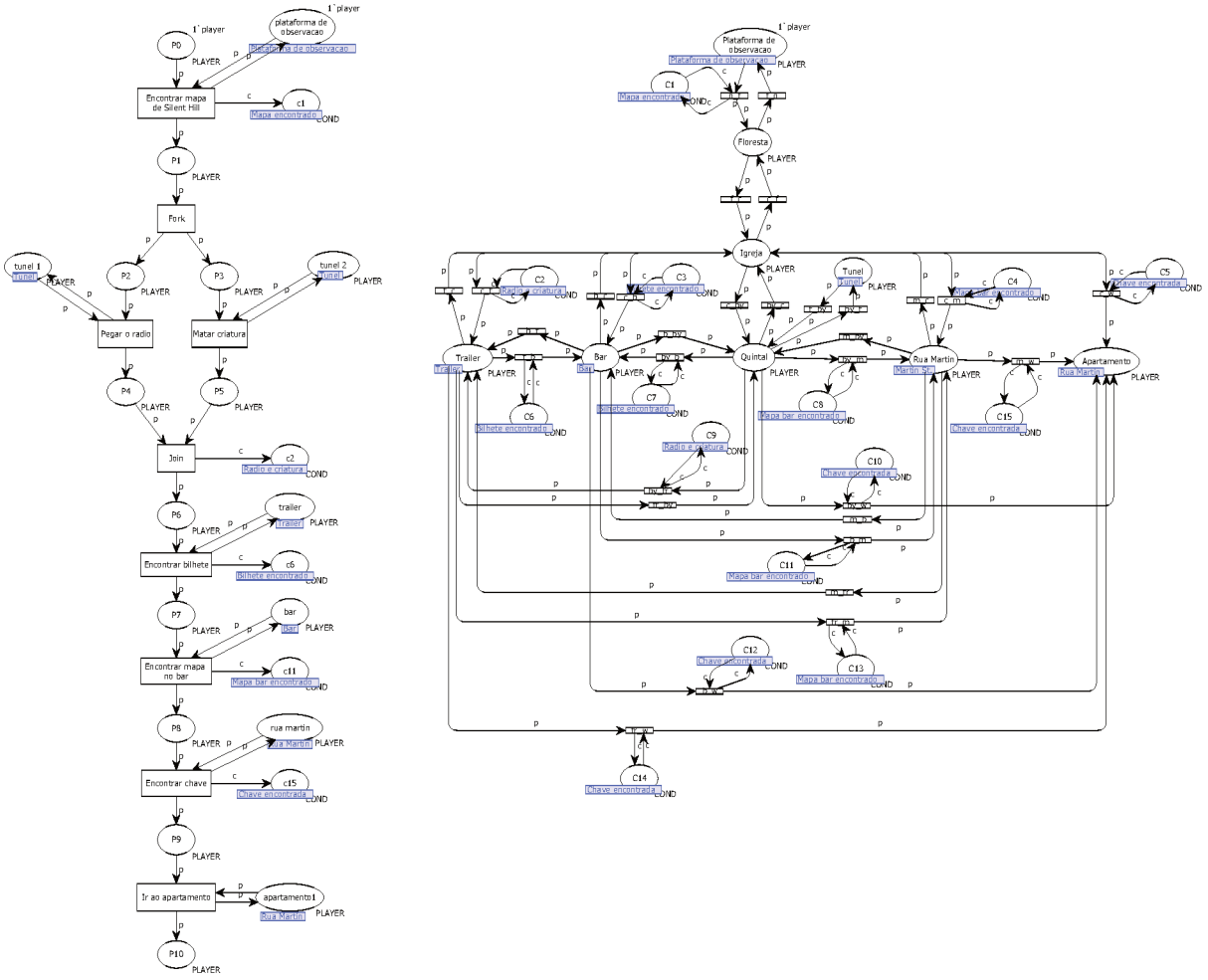


Figura 26 – Rede de Petri Colorida do Modelo Global do primeiro nível do jogo Silent Hill II.

transição extra t^* que conecta os lugares finais do Modelo Lógico e Modelo Topológico aos lugares de início dos mesmos modelos, como ilustrado na figura 27.

Definição 12 (Modelo de Análise) *O Modelo de Análise é dado por* $\overline{MG} = (\overline{ML}, \overline{MT}, \overline{ML})$ *onde:*

- ❏ $\overline{ML} = \boldsymbol{ML}$, *onde ML é o Modelo Lógico;*
- ❏ $\overline{MT} = \boldsymbol{MT}$, *onde MT é o Modelo Topológico;*
- ❏ $\overline{MC} = \boldsymbol{MC}$, *onde MC corresponde ao mecanismo de comunicação;*
- ❏ $\mathbf{t}^*$ *é uma transição extra que irá reiniciar o modelo conectando os lugares finais do Modelo Lógico e Modelo Topológico aos lugares de início dos mesmos modelos.*

Para ilustrar o Modelo de Análise, considere a figura 26 que ilustra o Modelo Global do primeiro nível de Silent Hill II. O lado esquerdo da figura representa o Modelo Lógico,

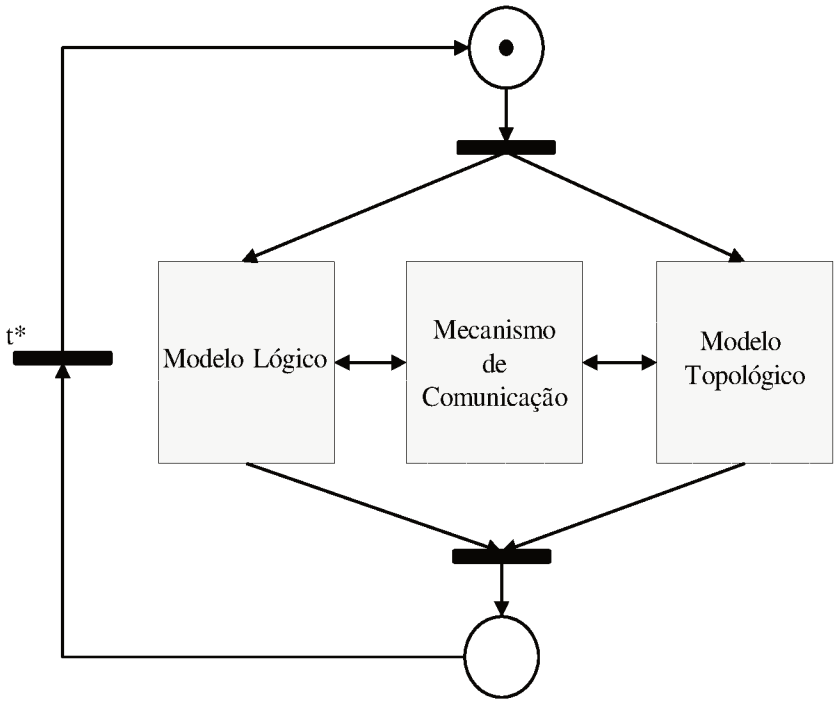


Figura 27 – Esquema para a construção do Modelo de Análise de jogo.

enquanto que o lado direito representa o Modelo Topológico. Para realizar a análise do Modelo Global, é adicionado um lugar de início comum, denominado de A1 e um lugar de fim, denominado de A2, conforme ilustra a figura 28.

Na figura 26 a transição *T1* é uma bifurcação que produz uma ficha no lugar *P0* (lugar de início do Modelo Lógico) e uma ficha no lugar *Plataforma de observação* (o lugar de início do mapa topológico, ou seja, o lugar que representa a área do mapa onde o jogador está inicialmente). A transição *T2* é uma junção que tem o propósito de consumir as fichas que (no caso *sound*) devem estar no lugar de fim do Modelo Lógico, no lugar que representa a área do mapa onde o jogador deve estar ao finalizar o nível, e nos lugares de condição marcados que são usados para liberar as passagens entre as áreas do mapa. Por fim a transição T3 representa a transição extra t^* que irá reiniciar o modelo, transformando-o assim em um modelo cíclico para que seja realizada a verificação das propriedades de vivacidade e limitabilidade. Verificando tais propriedades no modelo estendido cíclico, a propriedade *soundness* do modelo não cíclico é então garantida. Tal propriedade mostrará no caso de um jogo que cada atividade será executada por pelo menos um cenário de jogo e que todas as áreas do mapa topológico serão acessíveis no final do nível.

Para verificar se o modelo estendido $\overline{MG}$ é vivo e limitado é possível utilizar ferramentas de análise disponíveis no CPN Tools. O CPN Tools fornece duas abordagens para análise: simulação e espaço de estado (*state space*). A ideia básica por trás do *state space* é calcular todos os estados alcançáveis a partir da marcação inicial de uma rede de Petri, e

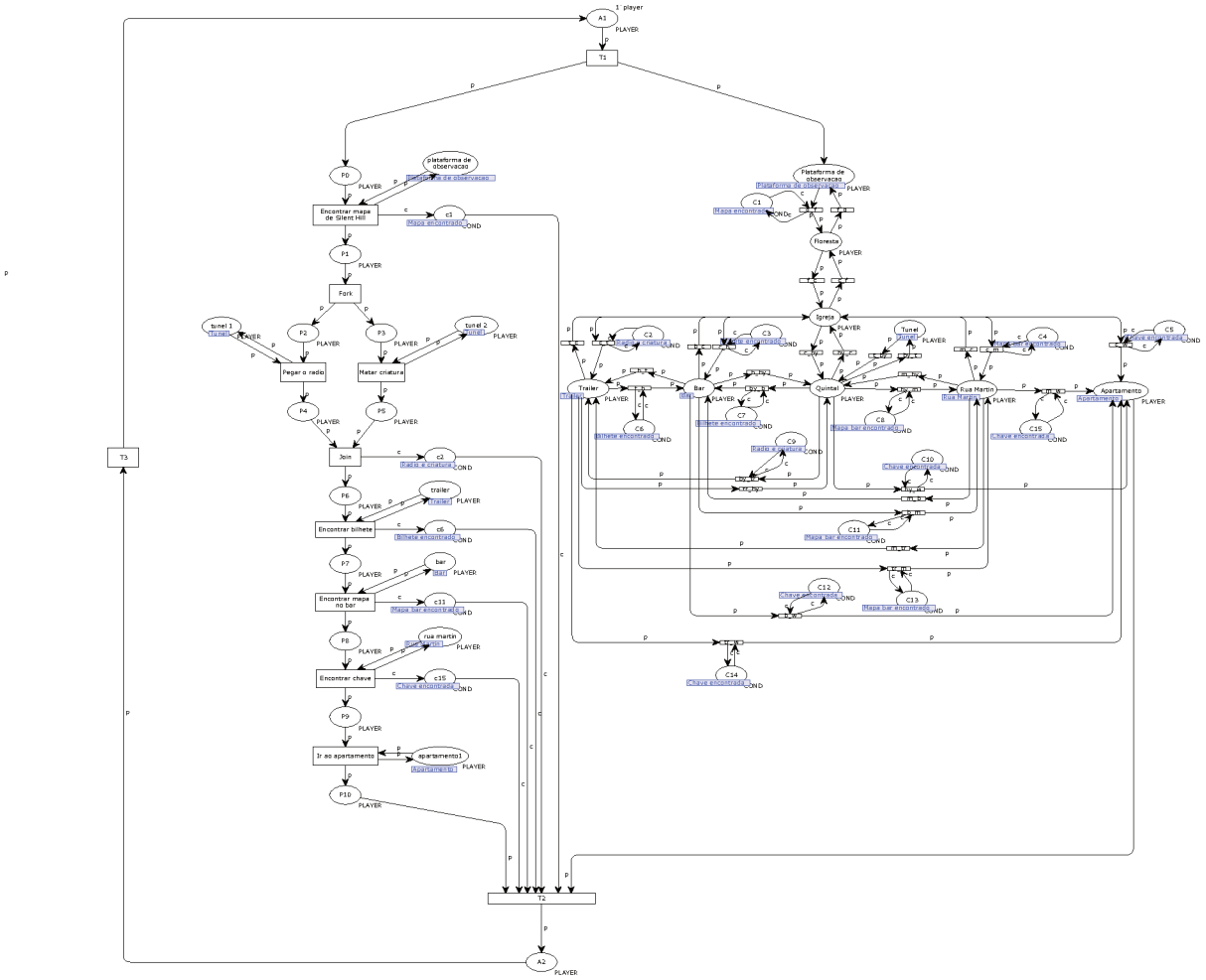
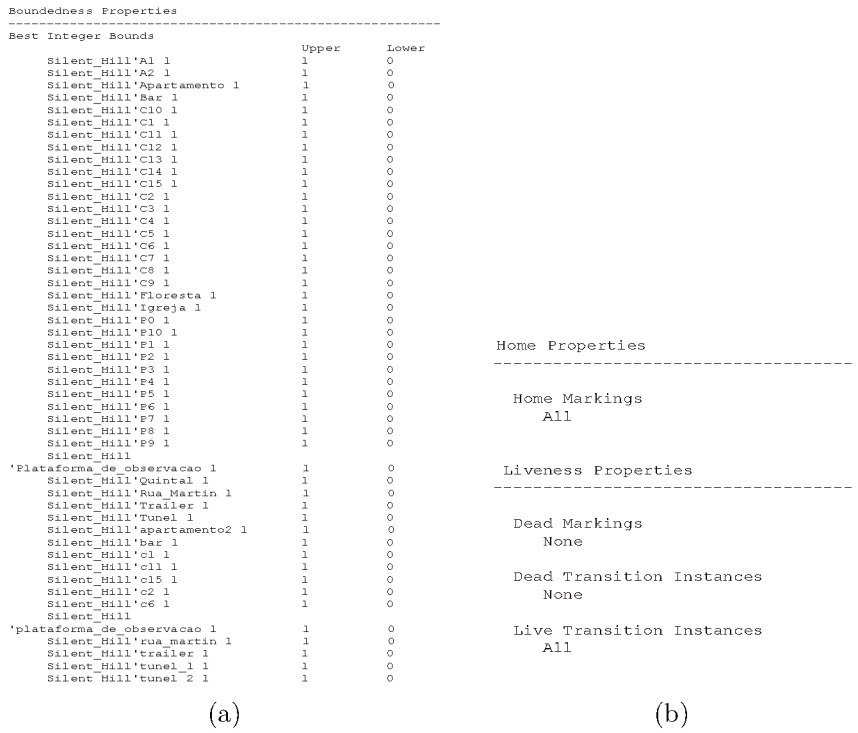


Figura 28 – Modelo de Análise do primeiro nível do jogo Silent Hill II.

representá-los em um grafo, onde os nós representam os estados alcançáveis e os arcos representam os possíveis disparos de transição. Algumas propriedades são então verificadas com base nos componentes fortemente conexas do grafo obtido. O grafo das marcações alcançáveis é gerado automaticamente e gera um relatório contendo as informações da análise.

Após a análise pelo *state space* da rede $\overline{MG}$ ilustrada na figura 28, o relatório gerado identifica que a rede é 1-limitada (figura 29(a)), ou seja, o número máximo de fichas que um lugar no modelo pode ter é 1 (não há duplicações de fichas). O relatório também indica que a rede analisada é viva (liveness), uma vez que todas as transições do modelo são vivas (figura 29(b)). Sendo assim, o modelo estendido $\overline{MG}$ é dito vivo e 1-limitado.

Se a rede $\overline{MG}$ é limitada (binária) e viva, de acordo com as definições dessas propriedades, então o modelo correspondente MG é considerado *sound*. Considerando o ponto de vista do jogo, um modelo *sound* significa que todas as atividades do nível poderão ser executadas e todas as áreas do mapa topológico poderão ser eventualmente acessadas pelo jogador. Isso garante a consistência das atividades do nível de jogo, bem como a boa



```
Boundedness Properties
------------------------------------------------------------
Best Integer Bounds
                                        Upper       Lower
     Silent_Hill'Al 1                   1           0
     Silent_Hill'A2 1                   1           0
     Silent_Hill'Apartamento 1          1           0
     Silent_Hill'Bar 1                  1           0
     Silent_Hill'C10 1                  1           0
     Silent_Hill'C1 1                   1           0
     Silent_Hill'C11 1                  1           0
     Silent_Hill'C12 1                  1           0
     Silent_Hill'C13 1                  1           0
     Silent_Hill'C14 1                  1           0
     Silent_Hill'C15 1                  1           0
     Silent_Hill'C2 1                   1           0
     Silent_Hill'C3 1                   1           0
     Silent_Hill'C4 1                   1           0
     Silent_Hill'C5 1                   1           0
     Silent_Hill'C6 1                   1           0
     Silent_Hill'C7 1                   1           0
     Silent_Hill'C8 1                   1           0
     Silent_Hill'C9 1                   1           0
     Silent_Hill'Floresta 1             1           0
     Silent_Hill'Igreja 1               1           0
     Silent_Hill'P0 1                   1           0
     Silent_Hill'P10 1                  1           0
     Silent_Hill'P1 1                   1           0
     Silent_Hill'P2 1                   1           0
     Silent_Hill'P3 1                   1           0
     Silent_Hill'P4 1                   1           0
     Silent_Hill'P5 1                   1           0
     Silent_Hill'P6 1                   1           0
     Silent_Hill'P7 1                   1           0
     Silent_Hill'P8 1                   1           0
     Silent_Hill'P9 1                   1           0
     Silent_Hill
'Plataforma_de_observacao 1             1           0
     Silent_Hill'Quintal 1              1           0
     Silent_Hill'Rua_Martin 1           1           0
     Silent_Hill'Trailer 1              1           0
     Silent_Hill'Tunel 1                1           0
     Silent_Hill'apartamento2 1         1           0
     Silent_Hill'bar 1                  1           0
     Silent_Hill'c1 1                   1           0
     Silent_Hill'c11 1                  1           0
     Silent_Hill'c15 1                  1           0
     Silent_Hill'c2 1                   1           0
     Silent_Hill'c6 1                   1           0
     Silent_Hill
'plataforma_de_observacao 1             1           0
     Silent_Hill'rua_martin 1           1           0
     Silent_Hill'trailer 1              1           0
     Silent_Hill'tunel_1 1              1           0
     Silent_Hill'tunel_2 1              1           0
```

(a)

```
Home Properties
------------------------------------

  Home Markings
     All


 Liveness Properties
------------------------------------

  Dead Markings
     None

  Dead Transition Instances
     None

  Live Transition Instances
     All
```

(b)

Figura 29 – Resultados da análise das propriedades de limitabilidade e vivacidade.

construção do mapa topológico.

4.2 Modelo de Simulação

Com modelos baseados em redes de Petri, é possível analisar e verificar a corretude de um sistema, ou ainda avaliar e estimar o seu desempenho. Uma das técnicas de análise amplamente utilizadas para esse fim é a de simulação. A simulação permite analisar o desempenho de um sistema e validar o modelo (AALST; STAHL, 2011). Ao simular um modelo de um vídeo game, por exemplo, é interessante saber os tempos de execução das atividades. Esse fator pode ser útil para medir o tempo médio de *quests* (missões) específicas que precisam ser executadas pelo jogador, ou ainda estimar o tempo médio do jogo como um todo.

Usar redes de Petri Coloridas como modelos de simulação é uma abordagem adequada, pois é possível considerar uma versão temporizada dos tipos de dados utilizados e anexar probabilidades a determinados eventos. Por exemplo, é possível simular a duração de uma determinada sequência de atividades, ou um caminho percorrido pelo jogador no mundo virtual. Usar softwares baseados em redes de Petri que dão suporte para a criação e simulação de tais modelos é essencial. Quando se trata de redes de Petri Coloridas, o software CPN Tools se apresenta como o mais completo em termos de funcionalidades. Em particular, por meio do CPN Tools, é possível criar os modelos e atribuir as probabilidades necessárias à geração de eventos aleatórios usando funções de distribuição aleatória

predefinidas na ferramenta. A ferramenta também disponibiliza meios para replicar os cenários e calcular os valores médios das partes relevantes e escolhidas do modelo colorido.

Nesta seção, será apresentada uma abordagem para a modelagem de cenários de vídeo games usando o formalismo das redes de Petri Coloridas com adição de tipos temporizados. Na seção 4.2.1, é apresentada a definição do Modelo Lógico Temporizado para descrever as durações de atividades de jogos. Na seção 4.2.2, é apresentado o Modelo Topológico Temporizado que descreve as áreas do mundo virtual e as durações de deslocamentos do jogador entre as diversas áreas do mapa. Na seção 4.2.3, é apresentado o Modelo Global Temporizado, formado pela junção do Modelo Lógico Temporizado e do Modelo Topológico Temporizado. Por meio do Modelo Global será então possível executar a simulação e, assim, estimar a duração média do jogo (ou de partes dele), como é apresentado na seção 4.2.4.

4.2.1 Modelo Lógico Temporizado

Em um vídeo game real, as atividades não são realizadas instantaneamente. Cada atividade de um jogo possui uma duração que é diferente de zero e que termina em uma quantidade finita de tempo. Portanto, um aspecto importante para se observar em um jogo é a duração média de suas atividades.

Cada nível do jogo possui uma duração média, que é a soma da duração de cada atividade daquele nível. Assim, no final de todos os níveis, é possível obter a duração média para a conclusão do jogo. É importante ressaltar que a duração de um jogo depende de alguns fatores como, por exemplo, o nível de experiência do jogador, a quantidade de atividades existentes no jogo, o grau de dificuldade de cada atividade, e até mesmo o número de vezes que uma atividade foi executada.

Uma vez que as atividades de um jogo podem ser representadas por meio de uma rede de Petri (como apresentado no capítulo 4), é possível então representar a duração de uma atividades por meio de funções associadas às atividades do modelo. Para tanto, é preciso assumir que cada atividade existente em um jogo leva uma quantidade de tempo (não nula) para ser executada. Além disso, também é necessário considerar uma função de distribuição de tempo que determinará a duração da atividade. Assim, o modelo resultante que representa as atividades de um nível de jogo e suas respectivas durações é denominado de Modelo Lógico Temporizado, e pode ser formalizado como se segue.

Definição 13 (Modelo Lógico Temporizado (ML^T)) *O Modelo Lógico Temporizado de um nível de jogo é dado por uma tupla (ML, Ω) onde:*

- ❑ ***ML*** *é um Modelo Lógico de nível de jogo;*

- ❑ *Ω é uma função que associa um número real não negativo ω a cada transição t_i do Modelo Lógico, tal que $\omega_i = \Omega(t_i)$ é denominado o tempo de execução da atividade t_i.*

Na definição 13, *ML* representa um Modelo Lógico de um nível de jogo baseado na definição 10. Ω é uma função de distribuição de tempo aleatório associada a cada transição que irá gerar um número real não negativo e diferente de zero para representar o tempo de execução de uma atividade. A execução de uma atividade t_i será então finalizada pelo jogador após o valor ω_i.

A função Ω pode ser definida por meio de uma função específica ou ainda por meio de uma das funções de distribuição aleatória existentes. Na abordagem apresentada nesta pesquisa, a função de tempo aleatório proposta para simular a duração que um jogador precisa para executar atividades específicas, é a função exponencial. A função exponencial é uma das funções de distribuição aleatória mais usadas para simular tempos entre eventos na maioria dos problemas de simulação (AALST; STAHL, 2011). A função é baseada em um parâmetro exponencial r, pertencente ao conjunto dos números reais ($\mathbb{R}$). Para um r positivo ($r > 0$), a função *exponencial(r)* produz um valor baseado na distribuição exponencial com média *1/r*.

O tempo de execução de uma atividade em um jogo depende de alguns fatores; um deles é a experiência do jogador. Uma determinada atividade pode ser considerada fácil por um jogador experiente, e ele levará então pouco tempo para executá-la. Já para um jogador com pouca experiência levará mais tempo para executar a mesma atividade. Assim, o parâmetro r da função exponencial para simular a duração de uma atividade é definido de acordo com o nível de experiência do jogador.

Na maioria dos jogos existem atividades que exigem que o jogador encontre objetos dispersos no mundo virtual, resolva enigmas, ou ainda interaja com personagens do próprio jogo, chamados de NPC (*Non-Player Character*). Muitas dessas interações podem ser propostas como desafios onde o jogador precisa ganhar uma competição, por exemplo. Alguns jogos permitem ainda que atividades desse tipo sejam executadas mais de uma vez, caso o jogador não consiga executar com sucesso na primeira tentativa. Dessa forma, existem atividades que correspondem à roteiros iterativos dentro do jogo. Como o tempo de execução de um nível de jogo é a soma dos tempos de execução de cada atividade, é importante levar em consideração o número de vezes que o jogador executa uma mesma atividade. Então se um jogador executar uma atividade no jogo e falhar, ele precisará repetir a mesma atividade até conseguir executá-la com sucesso. É razoável de se pensar que a cada iteração o jogador adquiri mais experiência de jogo e consequentemente poderá executar a atividade em menos tempo na próxima iteração. Portanto, em cada tentativa, o jogador aumenta o seu nível de experiência no jogo.

No primeiro nível do jogo Silent Hill II existe uma situação desse tipo. A atividade *Matar criatura* consiste em lutar contra uma criatura (um NPC). Se o jogador não vencer a

luta, ele terá que executar a atividade novamente, ou seja, o jogador lutará repetidamente contra a criatura até ele ser capaz de derrotá-la. No Modelo Lógico do primeiro nível de Silent Hill II, a atividade *Matar criatura* forma uma rotina paralela com a atividade *Pegar o rádio*. Assim, essas atividades podem ser executadas em qualquer sequência. Nesse caso em específico existem quatro opções de execução:

1. O jogador executa primeiro a atividade *Pegar o rádio* e obtém sucesso. Em seguida executa *Matar criatura*, também com sucesso, e segue para as próximas atividades do jogo.

2. O jogador executa primeiro a atividade *Matar criatura* e obtém sucesso. Em seguida executa *Pegar o rádio*, também com sucesso, e segue para as próximas atividades do jogo.

3. O jogador executa primeiro a atividade *Matar criatura* e perde. Como consequência, o jogo é reiniciado para o momento anterior que corresponde a conclusão da última atividade (*Encontrar mapa de Silent hill*).

4. O jogador executa primeiro a atividade *Pegar o rádio*, e então executa *Matar criatura* mas perde. Como consequência o jogador terá que voltar para a última atividade concluída (*Encontrar mapa de Silent hill*).

As opções 1 e 2 apresentam as situações em que o jogador consegue executar as atividades com sucesso, independente da ordem em que as executa. Já as opções 3 e 4 apresentam situações em que o jogador não consegue executar a atividade *Matar criatura* com sucesso, e então o jogo reiniciará e o jogador voltará para a última atividade concluída no jogo.

Na figura 30, as atividades *Pegar o rádio* e *Matar criatura* são representadas pelas transições de mesmo nome. A transição *L2* representa a opção 3 e a transição *L1* representa a opção 4. *L2* irá disparar quando existir uma ficha no lugar *P2* e uma ficha no lugar *P3* (opção 3). *L1* irá disparar se existir uma ficha em *P3*; se a transição *L1* é disparada então *L3* irá disparar se existir uma ficha no lugar *P11* e uma no lugar *P4* (opção 4). Esse modelo produz um tipo de rotina iterativa para o jogador quando ele falha em executar certas atividades.

Para representar a duração das atividades, uma função de tempo aleatório Ω é associada a cada transição do Modelo Lógico. A função exponencial é então utilizada em cada transição do modelo para simular a duração das atividades. O software CPN Tools implementa uma série de funções de distribuição aleatória. A função exponencial é uma dessas funções nativas da ferramenta. Portanto, para utilizá-la basta declarar a expressão *exponential(r)*, onde *r:real* e $r > 0$.

A figura 31 ilustra o Modelo Lógico Temporizado do primeiro nível do jogo Silent Hill II. A expressão *@++exponential(e)* é adicionada em cada transição do modelo para

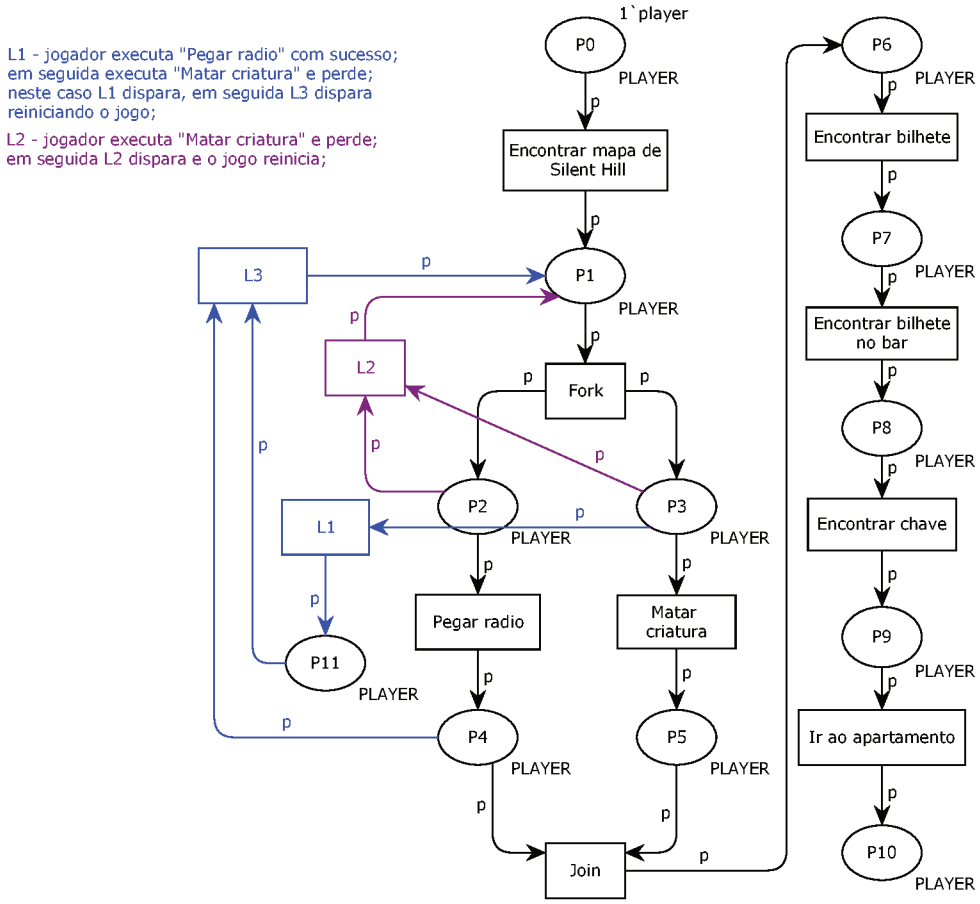


Figura 30 – Roteiro iterativo de atividades do Modelo Lógico do primeiro nível do jogo Silent Hill II.

determinar o tempo de duração da atividade. O parâmetro e da função exponencial representa o nível de experiência do jogador. Inicialmente esse parâmetro foi definido com o valor 0.5. Quando um jogador repete uma atividade, a experiência dele aumenta. Assim, a inscrição nos arcos de saída das transições *L2* e *L3* aumenta em 0.3 unidades o parâmetro e. Inicialmente os valores de experiência do jogador foram definidos apenas para fins de teste. É importante ressaltar que cada jogo requer um estudo mais preciso para definir o nível de experiência do jogador no início do jogo. Além disso, o nível de experiência pode variar de jogador para jogador.

Os lugares da rede da figura 31 pertencem ao tipo de dado *PLAYER*. A declaração dos tipos de dados para o modelo é dada da seguinte forma[2]:

```
1 colset REAL = real; (* tipo de dado que assume valores do conjunto dos
    numeros reais *)
2 colset P = with player; (* tipo de dado que assume o valor player *)
3 colset PLAYER = product P * REAL timed; (* produto entre os tipos de
    dados P e REAL *)
4 var p: P; (* variavel do tipo P *)
5 var e: REAL; (* variavel do tipo REAL *)
```

[2] O CPN Tools não suporta escrita com acentuação.

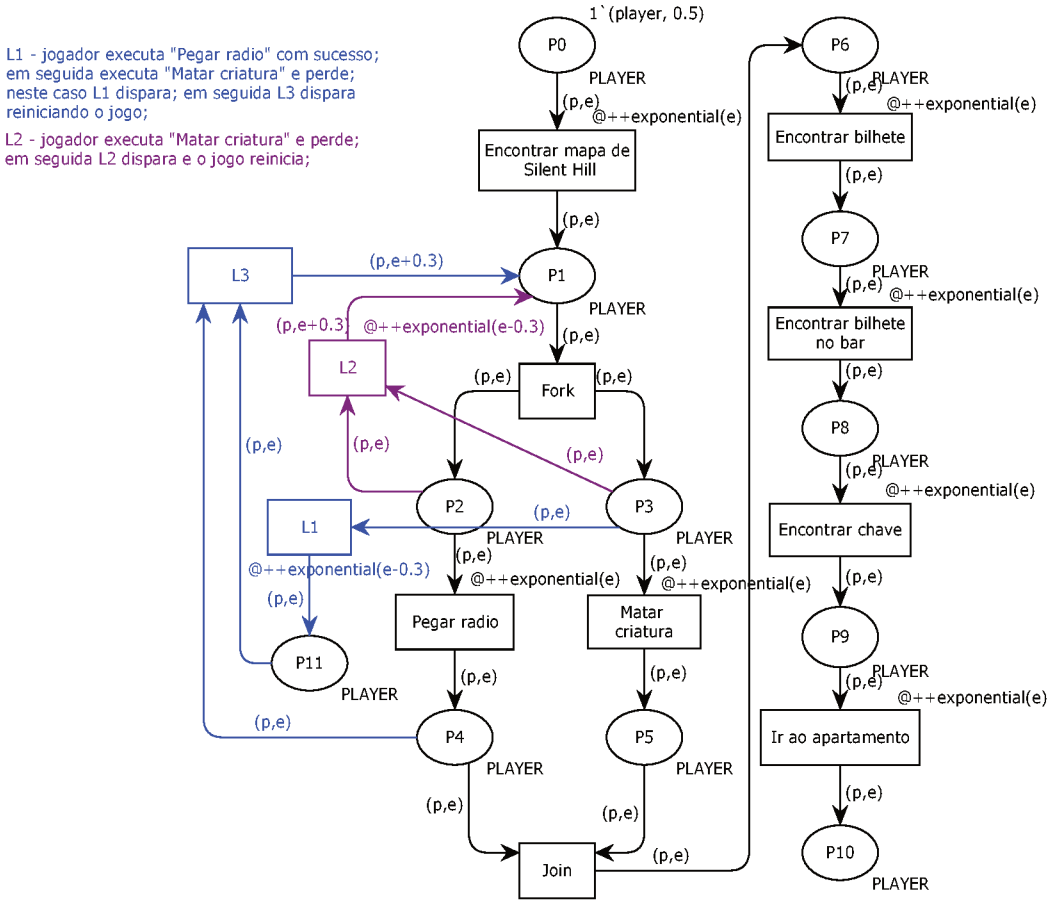


Figura 31 – Modelo Lógico Temporizado do primeiro nível do jogo Silent Hill II.

O *color set REAL* é um tipo primitivo de dado enquanto que o *color set P* é um tipo de dado específico que assume o valor *player*. O tipo *PLAYER* é um color set temporizado, definido com a palavra reservada *timed*, formado a partir do produto entre os tipos de dados *P* e REAL. Isso significa que o tipo de dado *PLAYER* pode executar simulações com tempo. Além disso, *PLAYER* é composto pelo produto dos *color sets P* e *REAL*. Dessa forma o conjunto de cores é dado por Σ = {REAL, P, PLAYER}. As variáveis *p* e *e* são definidas como sendo Tipo[p] = P e Tipo[e] = REAL, e são associadas a todos os arcos do modelo. Assim, a ficha no modelo que representa um jogador é dada pela expressão *1‘(player,0.5)*, ou seja, um jogador (*P*) com nível de experiência 0.5 (*REAL*).

4.2.2 Modelo Topológico Temporizado

O jogo precisa dar uma certa liberdade ao jogador em relação ao mundo virtual. O jogador precisa ter a possibilidade de se movimentar e interagir com objetos do jogo da forma que quiser. Ele pode ficar em uma área específica do mundo virtual por um curto período de tempo ou por um longo período de tempo. Além disso, o jogador pode se deslocar de uma área para a outra de forma rápida ou devagar e quantas vezes desejar.

O jogador terá então um tempo mínimo e um tempo máximo para se locomover entre

as áreas. No Modelo Topológico, o disparo de uma transição significa que o jogador se moveu de uma área para a outra. Para tanto, é preciso que cada transição do modelo possua uma função de distribuição de tempo para representar a duração da movimentação do jogador pelo mapa. É possível definir então um Modelo Topológico Temporizado de acordo com a definição 14.

Definição 14 (Modelo Topológico Temporizado (MT^T)) *Um Modelo Topológico Temporizado de jogo é dado por uma tupla (MT, Λ) onde:*

- ❏ ***MT*** *é um Modelo Topológico de jogo;*
- ❏ *Λ é uma função que associa um número real não negativo λ a cada transição t_i do Modelo Topológico, tal que $\lambda i_{min} = \Lambda(t_i)$ é denominado o tempo mínimo de movimentação do jogador entre as áreas conectadas por t_i, e $\lambda i_{max} = \Lambda(t_i)$ é denominado o tempo máximo de movimentação do jogador entre as áreas conectadas por t_i.*

Para simular o tempo de movimentação do jogador no Modelo Topológico, esta abordagem propõe o uso da função de distribuição uniforme associada a todas as transições do modelo. A função uniforme produz um número aleatório entre os parâmetros *a* e *b*, com probabilidade uniforme, ou seja, qualquer valor entre os parâmetros *a* e *b* possuem a mesma probabilidade. Assim como a função exponencial, a função uniforme é nativa do software CPN Tools e para usá-la basta usar a expressão *uniform(a,b)*. Os valores *a* e *b* pertencem ao conjunto dos números reais e desde que $b > a$, a função *uniforme(a,b)* irá produzir um valor a partir de uma distribuição uniforme com média *(a + b)/2* (AALST; STAHL, 2011).

A função de distribuição uniforme é adequada para representar a duração da movimentação do jogador no mapa topológico, pois possui um parâmetro mínimo (a) e um parâmetro máximo (b). Para exemplificar, considere o Modelo Topológico Temporizado do primeiro nível do jogo Silent Hill II apresentado na figura 32. A declaração dos tipos de dados para o modelo é dada da seguinte forma:

```
1 colset REAL = real; (* tipo de dado que assume valores do conjunto dos
     numeros reais *)
2 colset P = with player; (* tipo de dado que assume o valor player *)
3 colset MAPA = P timed; (* tipo de dado temporizado que assume o valor
     player *)
4 var p: P; (* variavel do tipo P *)
5 var a: REAL; (* variavel do tipo REAL *)
6 var b: REAL; (* variavel do tipo REAL *)
7 fun Time() = uniform(a,b); (* funcao de distribuicao uniforme com media
     (a + b)/2 *)
```

Os lugares da rede pertencem ao tipo de dado *MAPA*. O *color set MAPA* é temporizado e composto pelo *color set P*, que por sua vez é composto com valores *player*. Assim

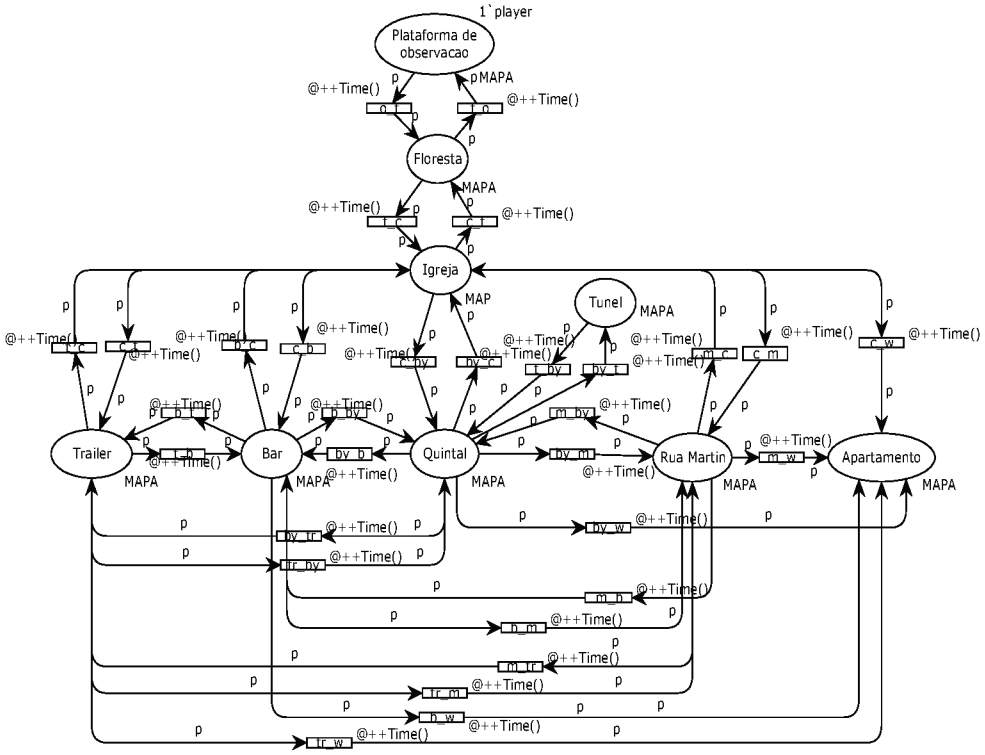


Figura 32 – Modelo Topológico Temporizado do primeiro nível do jogo Silent Hill II.

o conjunto de cores do modelo da figura 32 é formado por Σ = {REAL, P, MAPA}. As variáveis *p*, *a* e *b* são do tipo Tipo[p] = P, Tipo[a] = REAL e Tipo[b] = REAL, respectivamente. Como o Modelo Topológico do exemplo da figura 32 possui muitas transições, a fim de deixá-lo o mais claro possível, a função *Time()* é declarada. *Time()* é uma função que inclui a função *uniform(a,b)* e associa uma duração a cada ficha usada para disparar uma transição do Modelo Topológico.

Para o exemplo da figura 32, inicialmente os parâmetros *a* e *b* foram definidos como sendo $a = 0.1$ para a duração mínima, e $b = 1.0$ para a duração máxima. Dessa forma, o jogador irá levar entre 0.1 a 1.0 unidades de tempo para se locomover de uma área para a outra. Os parâmetros mínimo e máximo podem ser alterados. Para cada mundo virtual é possível então considerar parâmetros diferentes, pois a topologia do mapa muda.

4.2.3 Modelo Global Temporizado

Uma vez que o Modelo Lógico Temporizado e Modelo Topológico Temporizado estão estabelecidos, um Modelo Global Temporizado de nível de jogo precisa ser definido a fim de estabelecer as interações que um modelo tem sobre o outro. Por meio de mecanismos de comunicação definidos, as interações entre ambos modelos podem ser estabelecidas. Por se tratar de modelos temporizados, o Modelo Global Temporizado de um nível pode ser definido formalmente de acordo com a definição abaixo.

Definição 15 (Modelo Global Temporizado (MG^T)) *O Modelo Global Temporizado*

de jogo é uma tupla (ML^T, MT^T, MC) onde:

- ❏ ***ML^T*** *é uma rede de Petri Temporizada que representa o Modelo Lógico;*
- ❏ ***MT^T*** *é uma rede de Petri Temporizada que representa o Modelo Topológico;*
- ❏ ***MC*** *é uma rede de Petri que representa os Mecanismos de Comunicação.*

O modelo ML^T irá representar as atividades do jogo, cada uma com seu tempo estimado de duração. O modelo MT^T irá representar o mapa topológico do jogo e as restrições sobre as durações para se locomover entre as áreas do mapa. E o modelo MC mostra as interações entre os mecanismos de comunicação assíncrona que conectam os dois modelos. O exemplo da figura 33 ilustra o Modelo Global Temporizado do primeiro nível do jogo Silent Hill II.

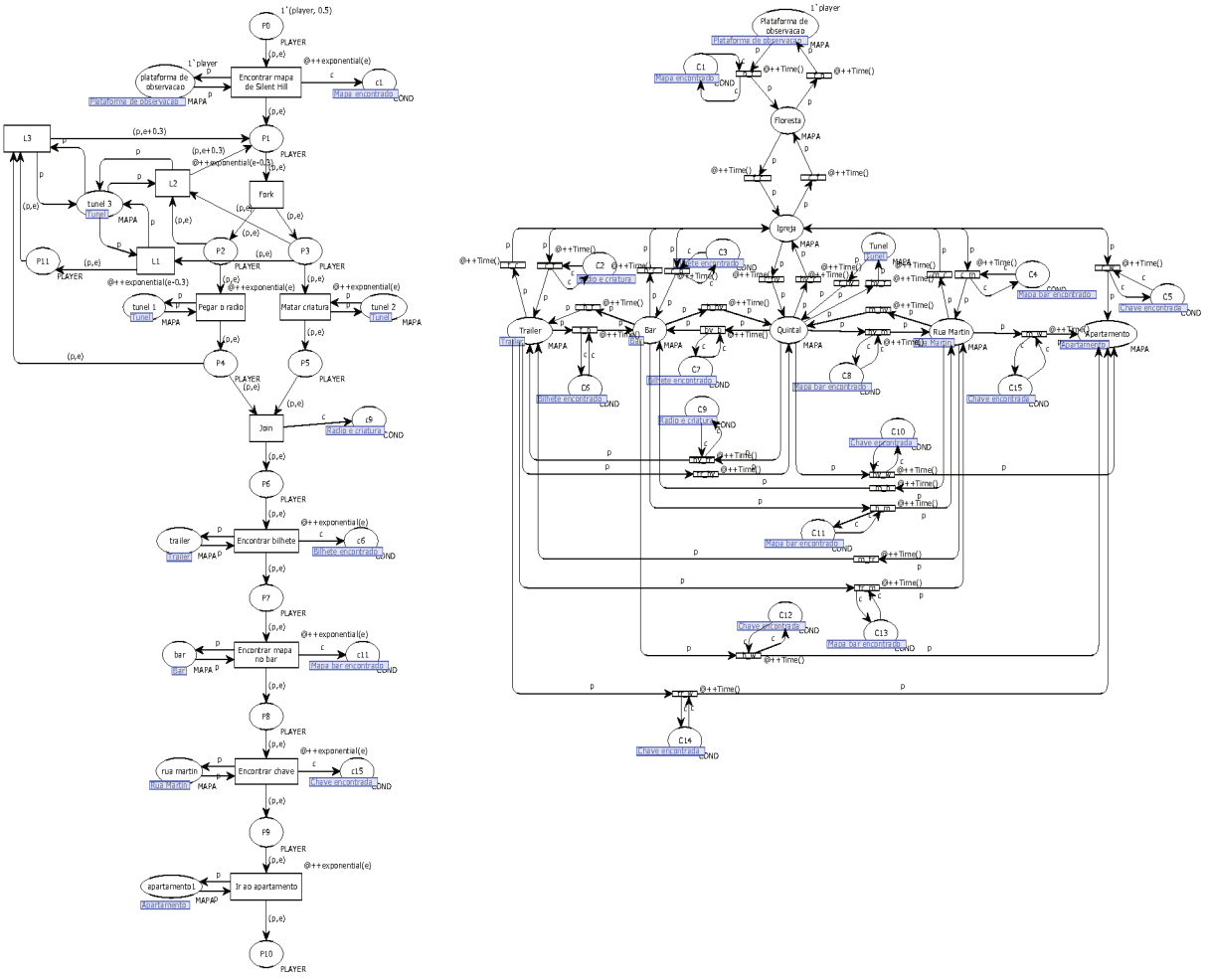


Figura 33 – Rede de Petri Colorida do Modelo Global do primeiro nível do jogo Silent Hill II.

4.2.4 Simulação do Modelo Global Temporizado

O processo de simulação faz parte da validação de sistemas. Uma simulação pode ser utilizada para explorar um número finito de execuções do sistema em questão. Assim, a validação de um sistema por simulação pode ser adequada para detectar erros e para obter um aumento de confiança em relação à corretude do sistema (JENSEN; KRISTENSEN, 2009). O simulador do CPN Tools permite dois tipos de simulação: automática e interativa.

Uma simulação interativa fornece uma maneira de investigar diferentes cenários em detalhes e verificar como o modelo funciona. Este modo também permite ao usuário escolher uma transição e as atribuições das variáveis. No modo automático, a simulação pode ser realizada por replicação ou no modo de avanço rápido. O final de uma simulação é determinado pelos critérios de parada da simulação. Na simulação automática por replicação, um relatório de simulação é produzido com os dados de cada replicação realizada.

Nesta abordagem, a simulação automática por replicação é proposta para estimar a duração média de um nível de jogo. Todas as simulações apresentadas aqui são realizadas automaticamente 10 vezes. Portanto, uma funcionalidade de replicação do CPN Tools, expressa pela inscrição *CPN'Replications.nreplications 10*, é utilizada. A quantidade de replicação pode variar. De acordo com Rozinat et al. (2008), quanto maior o número de simulações, melhor. Geralmente, a partir de um certo número de replicações, é possível perceber experimentalmente que o acréscimo de novas replicações não acrescenta nenhuma informação útil aos valores médios considerados pelo problema estudado.

Cada modelo criado (lógico e topológico) pode ser simulado separadamente. Por exemplo, o modelo da figura 31 que representa o Modelo Lógico Temporizado do primeiro nível de Silent Hill II foi submetido a simulação automática com 10 replicações.

A figura 34 apresenta o relatório gerado ao final da simulação. O relatório gera para cada simulação o número de passos e o tempo de execução do modelo até um critério de parada. Por se tratar de um Modelo Lógico de nível de jogo, o critério de parada é não ter mais transições habilitadas para disparo, ou seja, a sequência de atividades foi executada a partir do momento inicial do nível. É possível identificar no relatório da figura 34 que todas as simulações foram finalizadas por não haver mais transições habilitas ("*Stop reason...: No more enable transitions!*"). O número de passos para finalizar as simulações do modelo pode variar de simulação para simulação. Por exemplo, na simulação número 1, o número de passos foi 9 ("*Steps.....: 9*"). Já na simulação número 5 foram necessários 17 passos para que a execução fosse finalizada ("*Steps.....: 17*"). Esse comportamento é normal, pois o modelo possui roteiros iterativos que representam a possibilidade do jogador executar várias vezes algumas atividades. O tempo do modelo também é variável entre as simulações. Por exemplo, a duração mínima para executar todas as tarefas do nível simulado corresponde a aproximadamente 8,54 unidades de tempo ("*Simulation no.: 1*").

Já a duração máxima para executar todas as atividades corresponde a aproximadamente 23,41 unidades de tempo ("*Simulation no.: 5*"). Com todos os tempos gerados pelas simulações, é possível calcular que a média para executar todas as atividades do nível corresponde a aproximadamente 13,7 unidades de tempo.

```
Simulation no.: 1
Steps.........: 9
Model time....: 8.54950963137
Stop reason...: No more enabled transitions!
Time to run simulation: 0 seconds

Simulation no.: 2
Steps.........: 9
Model time....: 15.30329904
Stop reason...: No more enabled transitions!
Time to run simulation: 0 seconds

Simulation no.: 3
Steps.........: 9
Model time....: 9.77368666598
Stop reason...: No more enabled transitions!
Time to run simulation: 0 seconds

Simulation no.: 4
Steps.........: 15
Model time....: 17.0892393767
Stop reason...: No more enabled transitions!
Time to run simulation: 0 seconds

Simulation no.: 5
Steps.........: 17
Model time....: 23.4159232825
Stop reason...: No more enabled transitions!
Time to run simulation: 0 seconds

Simulation no.: 6
Steps.........: 9
Model time....: 13.9048727582
Stop reason...: No more enabled transitions!
Time to run simulation: 0 seconds

Simulation no.: 7
Steps.........: 15
Model time....: 8.68495576436
Stop reason...: No more enabled transitions!
Time to run simulation: 0 seconds

Simulation no.: 8
Steps.........: 13
Model time....: 15.4814450027
Stop reason...: No more enabled transitions!
Time to run simulation: 0 seconds

Simulation no.: 9
Steps.........: 9
Model time....: 15.6245852868
Stop reason...: No more enabled transitions!
Time to run simulation: 0 seconds

Simulation no.: 10
Steps.........: 9
Model time....: 9.91201704388
Stop reason...: No more enabled transitions!
Time to run simulation: 0 seconds
```

Figura 34 – Relatório de simulação do Modelo Lógico Temporizado.

O Modelo Topológico Temporizado também pode ser simulado para estimar o tempo médio para percorrer todas as áreas do mapa. O modelo da figura 32 foi então submetido a simulação automática com 10 replicações. A figura 35 ilustra o relatório de simulação do Modelo Topológico Temporizado. O número de passos pode variar entre as simulações, pois o jogador pode ir livremente de uma área para outra e quantas vezes quiser. Para todas as simulações, o critério de parada apresentado foi a falta de transições habilitadas (*"Stop reason...: No more enable transitions!")*. De acordo com o modelo da figura 32, isso significa que o jogador saiu do lugar inicial (*Plataforma de observação*) e chegou até o lugar final do nível (*Apartamento*).

O relatório da figura 35 mostra que com 10 replicações foram necessários no mínimo 4 passos e no máximo 18 passos para fazer o percurso do lugar de início até o lugar de fim. O tempo máximo para realizar o percurso foi de aproximadamente 10,50 unidades de tempo ("*Simulation no.: 6*"), enquanto que o tempo mínimo foi de aproximadamente 1,17 unidades de tempo ("*Simulation no.: 2*"). O tempo médio para percorrer as áreas do mapa topológico então é de aproximadamente 5 unidades de tempo.

Em um jogo, o jogador só pode executar uma atividade se estiver na área apropriada. E algumas áreas são acessadas somente após a execução de uma atividade específica. Dessa forma, um nível é então composto pelas atividades e pelas áreas mundo virtual. É necessário então manter ambos modelos (lógico e topológico) juntos para estimar uma duração realista do jogo.

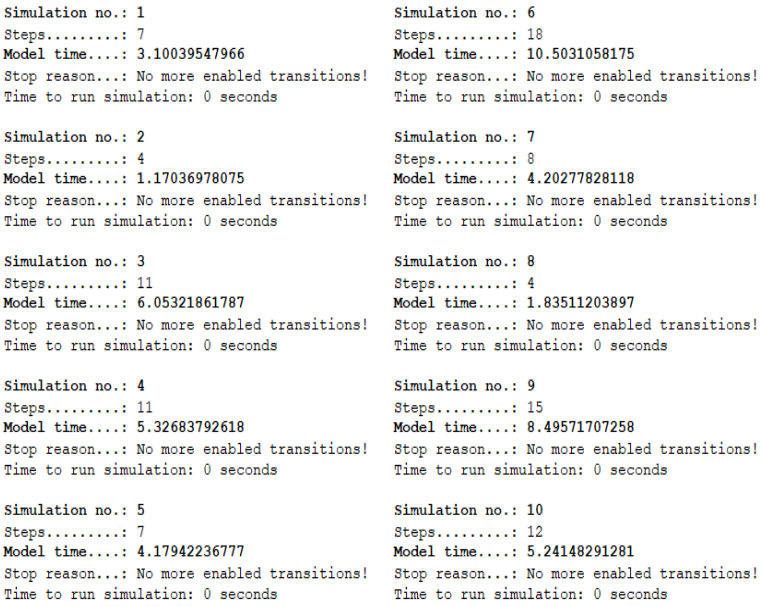


Figura 35 – Relatório de simulação do Modelo Topológico Temporizado.

Para estimar a duração do primeiro nível de Silent Hill II, o Modelo Global Temporizado apresentado na figura 33 é simulado. De acordo com o relatório de simulação apresentado na figura 36, a duração mínima para completar o nível corresponde a aproximadamente 59,13 ("*Simulation no.: 3*") unidades de tempo, e a duração máxima corresponde a aproximadamente 210,44 unidades de tempo ("*Simulation no.: 6*"). A duração média do nível corresponde a aproximadamente 87,16 unidades de tempo.

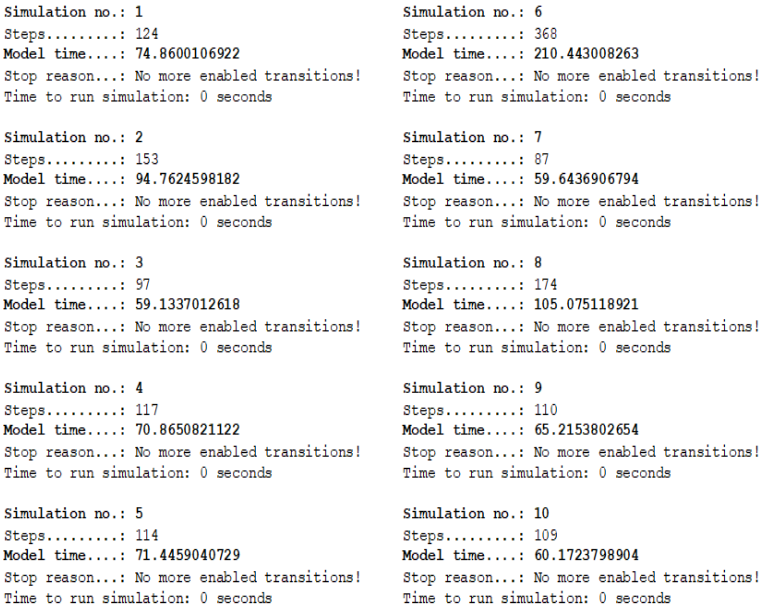


Figura 36 – Relatório de simulação do Modelo Global Temporizado.

Quando se considera a simulação dos modelos separadamente, a duração estimada refere-se apenas à propriedade desse modelo específico. Isso não acontece quando se

considera o Modelo Global, porque um modelo influencia o outro. Por exemplo, quando o Modelo Topológico é simulado sozinho, todas as áreas do mapa podem ser acessadas. Isso não é possível quando o Modelo Global é simulado, porque o jogador precisa executar tarefas para obter acesso a algumas áreas. Quando o Modelo Lógico é simulado sozinho, todas as atividades podem ser executadas quase imediatamente, o que novamente não é possível quando o Modelo Global é simulado. De fato, o jogador precisa estar em uma área específica do mapa para executar uma atividade específica do jogo. Dessa forma, os dois modelos precisam ser considerados juntos por meio dos mecanismos de comunicação a fim de produzir uma estimação realista do tempo de jogo dentro de um nível específico.

Capítulo 5

Modelagem de Vídeo Games baseado em Redes de Petri no caso Multiplayer

De acordo com Zagal, Nussbaum e Rosas (2000), a grande maioria dos jogos jogados em todo o mundo são de natureza coletiva. Apesar disso, os primeiros jogos eletrônicos eram do tipo singleplayer, devido a escassez e alto custo das tecnologias existentes. Com a evolução das tecnologias ao longo dos anos, os jogos passaram de uma simples aplicação com poucos elementos para sistemas complexos com a presença de uma narrativa bem definida e cenários cinematográficos. Após anos de evolução dos jogos é possível identificar a presença de vários gêneros distribuídos, geralmente, entre duas categorias: jogos singleplayer (único jogador) e jogos multiplayer (vários jogadores).

Na categoria singleplayer, existe apenas um jogador para interagir com todos os objetos do jogo e executar todas as atividades a fim de concluir o objetivo da narrativa. Por outro lado, na categoria multiplayer, as atividades e os objetos do jogo são sempre compartilhados entre 2 ou mais jogadores. Isso faz com que as ações de cada jogador impacte no mundo virtual do jogo e influencie, de certa forma, as ações dos demais jogadores. Por exemplo, se um dos jogadores recupera um objeto único do jogo (como uma chave), então esse objeto não ficará mais disponível para os demais. Consequentemente, isso pode impedir que os outros jogadores executem uma atividade específica do jogo que depende exclusivamente do objeto não recuperado (como abrir uma porta, por exemplo).

Portanto, um jogo multiplayer é diferente de um jogo singleplayer, sendo então importante encontrar e definir as características fundamentais dessa categoria, buscando entender como os elementos do jogo se relacionam entre si. De acordo com Araújo e Roque (2009), a representação de simulação de cenários multiplayer é um desafio na criação de vídeo games, pois em jogos com mais de um jogador, a dinâmica dos eventos concorrentes pode representar modelos de cenários muito complexos. Além disso, os jogos consistem de um conjunto de sistemas que interagem entre si e que precisam reagir a qualquer entrada possível dos jogadores de uma maneira bem definida (REUTER; GÖBEL; STEINMETZ, 2015). A modelagem de cenários multiplayer utilizando as redes de Petri é

efetiva para a simulação e identificação de erros nos estágios iniciais do desenvolvimento de jogos, uma vez que as redes de Petri são expressivas, fáceis de entender e possuem capacidade de modelar operações concorrentes (ARAÚJO; ROQUE, 2009).

Neste capítulo será apresentado um método, baseado nas redes de Petri do tipo Work-Flow net Possibilística, grafo de estados e redes de Petri Coloridas, para a modelagem e análise de cenários de vídeo games multiplayer.

5.1 Modelo de Análise

5.1.1 Modelo Lógico Multiplayer

Em geral, jogos multiplayer possuem três modos de execução: cooperativo, PvP (*Player vs Player*) e *deathmatch* (conhecido também como "mata-mata" pela comunidade brasileira de jogos). Os jogos com modo cooperativo permitem que os jogadores atuem como uma equipe para alcançar objetivos em comum. Neste tipo de modalidade, os jogadores podem ajudar uns aos outros. O modo PvP permite conflitos interativos no jogo entre dois ou mais participantes. Apesar de permitir a competição entre os usuários, jogos em modo PvP também podem permitir em certos momentos a cooperação entre os jogadores (na formação de equipes, por exemplo). Já um jogo multiplayer com modo *deathmatch* é organizado em partidas onde o objetivo é executar o máximo de jogadores possível até se alcançar um limite específico, seja o número de jogadores abatidos ou o tempo da partida. Quando o limite é alcançado, a partida do jogo termina e o vencedor é aquele jogador que mais acumulou pontos.

Apesar de diferentes modos, o conceito de cenário nos jogos multiplayer pode ser entendido como a sequência lógica e ordenada de atividades (eventos). A sequência das atividades dependem diretamente dos objetivos do jogo. Esses objetivos serão alcançados pelos jogadores por meio da execução ordenada da sequência de atividades estabelecida. Em jogos multiplayer, algumas dessas atividades podem (em alguns casos, devem) ser realizadas com a participação de outros jogadores. Dessa forma, jogos dessa categoria possuem um alto nível de interação social fazendo com que o jogo seja finalizado somente se existir algum tipo de interação entre os participantes. Jogos multiplayer em modo PvP ou Cooperativo geralmente possuem uma narrativa mais elaborada e, consequentemente, possuem uma sequência lógica das atividades mais complexa. Os jogos em modo *deathmatch* permitem a cada partida que os usuários interajam entre si por meio de ações limitadas e sempre com o mesmo objetivo, que é o de derrotar os oponentes.

A representação de cenários de jogos singleplayer usando WorkFlow nets é viável, como apresentado no capítulo 4. Nesse cenário, um único jogador existe e, consequentemente, executa todas as atividades necessárias para alcançar os objetivos do jogo sem interrupções de eventos externos. Em cenários com mais de um jogador, os jogadores precisam coexistir

e isso implica em compartilhar atividades e interferir diretamente ou indiretamente na execução das atividades de outros jogadores. Por exemplo, considere que uma atividade de um jogo seja coletar uma chave para que em seguida se abra uma porta. Ao passar por essa porta o jogador poderá seguir para várias áreas do jogo e executar outras atividades. No entanto, após um certo tempo, a porta se tranca automaticamente. Em um cenário com 3 jogadores, em que 1 deles coleta a chave e passa pela porta sozinho, 2 jogadores ficarão impossibilitados de seguir o mesmo caminho e executar as demais atividades. Os dois jogadores teriam então que esperar o detentor da chave retornar para abrir a porta novamente. Isso pode levar um tempo muito grande para acontecer, ou talvez, nunca acontecer. Uma forma de resolver tal problema seria então que os 3 jogadores compartilhassem essas atividades juntos. Se, neste mesmo exemplo, for considerado que a porta não se fecha automaticamente, então as atividades de "coletar a chave" e "destrancar a porta" só poderão ser executadas uma única vez, e uma vez executadas por um jogador, as atividades deverão ser executadas por todos os outros jogadores.

Ambas situações descritas acima geram um evento inesperado devido a interação de um jogador e podem introduzir incertezas na execução das atividades. Tais incertezas devem ser consideradas no Modelo Lógico Multiplayer a fim de tornar o modelo que representa as atividades do jogo mais consistentes entre o Modelo Lógico e a execução real do jogo. Para lidar com o comportamento das interações diretas entre os jogadores é possível utilizar um Modelo Lógico baseado em um tipo especial de rede de Petri denominado de WorkFlow net Possibilística. A rede em questão foi criada com a junção das redes de Petri Possibilísticas e da estrutura de roteamento das WorkFlow nets (REZENDE; JULIA; CARDOSO, 2012). Um modelo baseado em WorkFlow nets Possibilísticas possui a capacidade de se adaptar quando eventos inesperados acontecem por meio da noção de incerteza e de disparos incertos. Tais conceitos permitem então a modelagem de cenários de jogos quando existe a interação de 2 ou mais jogadores ao mesmo tempo compartilhando ou disputando certas atividades. A definição do Modelo Lógico para um jogo *multiplayer* é apresentada a seguir.

Definição 16 (Modelo Lógico Multiplayer) *Um Modelo Lógico Multiplayer (MLM) é uma rede de Petri que representa o sequenciamento lógico das atividades de um jogo multiplayer, e é dada pela WorkFlow net Possibilística (C_{aso}, P, T, V_f, Pré, Pós, A_{tc}, A_{ta}, M_0), tal que:*

- *"C_{aso}" representa a classe do objeto do tipo "Jogador". Um conjunto de atributos é definido para essa classe e organizado em hierarquia.*
- *V_f é o conjunto finito de variáveis formais do tipo "Jogador".*
- *A_{tc} é uma aplicação que, para cada transição $t \in T$, associa uma função de autorização η que envolve o conjunto de atributos dos objetos por meio das variáveis*

formais associadas aos arcos de entrada de t.

- A_{ta} *é uma aplicação que, para cada transição* $t \in T$*, associa uma ação que envolve os atributos das variáveis formais associados aos arcos de entrada, permitindo modificar seus atributos.*

- M_0 *é a marcação inicial que associa ao lugar de início uma soma formal dos objetos (instâncias da classe "Jogador") superior ou igual a 2 no caso multiplayer. A marcação inicial da rede* M_0 *sempre será no lugar de início*

Para o Modelo Lógico no caso multiplayer, também é importante considerar os conceitos de tipos de marcação e regras de disparo definidos para uma WorkFlow net Possibilística, ambos apresentados no capítulo 2, na seção 2.3 deste documento. Os tipo de marcação são categorizados em marcação precisa (cada objeto está localizado em apenas um lugar) e marcação imprecisa (um mesmo objeto está em dois ou mais lugares). Quanto as regras de disparo, há de se considerar o disparo certo (objetos são removidos dos lugares de entrada de uma transição e adicionados nos seus lugares de saída) e o disparo incerto ou pseudo-disparo (objetos são adicionados nos lugares de saída de uma transição, mas não são removidos dos seus lugares de entrada).

Para ilustrar o modelo MLM proposto, será usado o vídeo game Tom Clancy's Ghost Recon: Wildlands (UBISOFT, 2017). Ghost Recon é um jogo multiplayer cuja narrativa se passa na Bolívia, na América do Sul. Os jogadores são enviados para trás das linhas inimigas para criar situações que desestabilizarão e quebrarão a aliança entre o cartel de drogas mexicano de Santa Blanca e o governo corrompido do país. O jogo tem uma variedade de missões (*quests*) complexas que podem ser executadas em qualquer ordem pelos jogadores. Cada missão tem um objetivo específico. A conclusão de uma missão contribuirá ao objetivo do jogo, que é destruir o cartel de Santa Blanca.

O jogo é multiplayer com modo cooperativo, permitindo de um até quatro jogadores ativos ao mesmo tempo. O jogo possui várias *quests* distribuídas e uma delas é a *quest* chamada *The Chemist*. O objetivo é entrar na área das forças inimigas e resgatar um NPC (*Non Player Character*) conhecido como *Químico*. Para tanto, os jogadores precisam executar a seguinte sequência de atividades:

1. invadir a área (T0);
2. enfrentar os inimigos (T1);
3. usar uma estratégia para escapar dos inimigos (T2);
4. encontrar o Químico na área correspondente (T5);
5. capturar o Químico com vida (T6);
6. transportar o prisioneiro para a área indicada no mapa (T7);

7. finalizar a missão (T8).

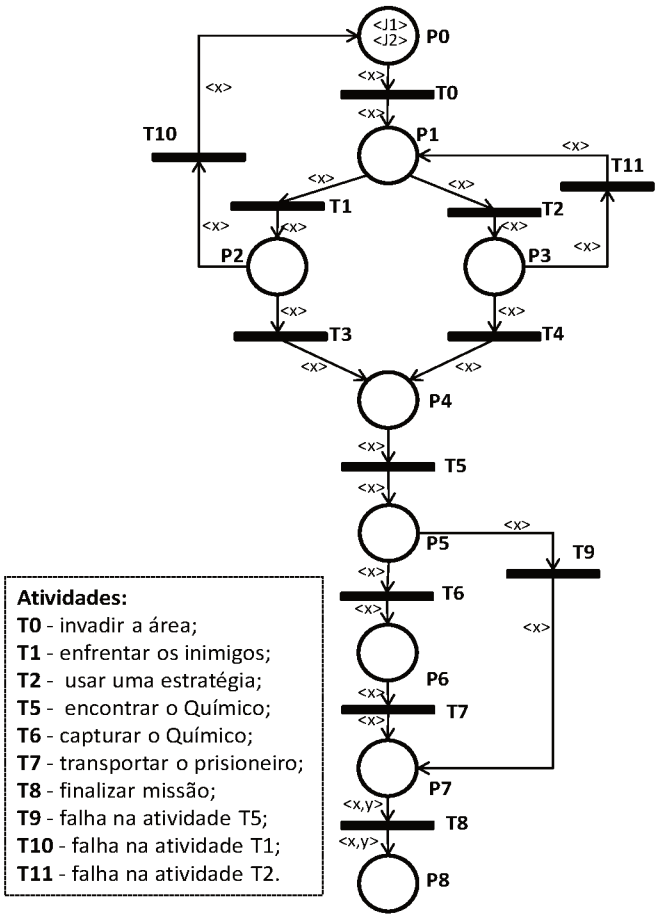


Figura 37 – Modelo Lógico Multiplayer da *quest The Chemist* do jogo Tom Clancy's Ghost Recon: Wildlands.

A figura 37 ilustra o Modelo Lógico Multiplayer da *quest The Chemist*. No modelo da figura 37, o lugar *P0* representa o início da *quest*, enquanto que o lugar *P8* representa o fim da *quest*. As transições de *T0* até *T8* representam as atividades da *quest*. Cada *quest* de jogo possui suas particularidades de narrativa. Por exemplo, em *The Chemist*, as atividades *T1* e *T2* pertencem a um roteiro condicional, ou seja, os jogadores podem escolher entre uma atividade ou outra. Portanto, *T1* e *T2* formam um *or-split* e *T3* e *T4* formam um *or-join* (AALST, 1998). As transições *T5*, *T6*, *T7* e *T8* formam uma rotina sequencial, o que significa que as atividades são executadas em sequência. Nessa *quest*, se o jogador falhar em executar alguma atividade, é possível voltar ao estado anterior e executar a atividade novamente. Para representar essa particularidade são usadas as transições *T9*, *T10* e *T11*. Considere a atividade *Enfrentar os inimigos*, representada pela transição *T1* na figura 37; se o jogador executa esta atividade com sucesso, então a transição *T3* será disparada e o jogador poderá seguir adiante no jogo. Caso contrário, se o jogador falhar, então a transição *T10* será disparada e o jogador retornará ao início

da *quest*. Esse tipo de situação corresponde a uma estrutura de rotina iterativa em uma WorkFlow net (AALST, 1998).

Se existir apenas um jogador, a execução das atividades acorrerá de acordo com o fluxo definido pelo modelo. No entanto, com mais de um jogador interagindo ao mesmo tempo, podem acontecer mudanças na execução do fluxo de atividades. Isso se deve ao fato das particularidades da narrativa dessa *quest*. Por exemplo, a atividade *T6* só pode ser executada uma única vez. Se essa atividade for executada por um jogador, nenhum outro jogador ativo no jogo poderá executá-la novamente. Portanto, se *T6* for executada por um jogador, a atividade estará implicitamente executada pelos outros jogadores ativos. Por fazerem parte de uma rotina sequencial com a atividade *T6*, as atividades *T5*, *T7* e *T8* também só poderão ser executadas uma única vez. Esse tipo de comportamento, se executado por uma rede de Petri ordinária (somente disparos certos), provocará uma inconsistência entre a execução do Modelo Lógico simulado e o comportamento real do grupo de jogadores.

Na figura 37, *J1* e *J2* representam os objetos pertencentes à classe "Jogador". Todos os lugares do modelo também pertencem à mesma classe, assim como as variáveis x e y que estão associadas aos arcos do modelo. *J1* e *J2* representam dois jogadores. Os atributos definidos para a classe nessa *quest* são todos booleanos[1] e são descritos por: *falha1*, *falha2*, *falha3* e *fim*. Os atributos *falha1* e *falha2*, quando verdadeiros, representam o fato do jogador ter falhado ao executar as atividades *Enfrentar os inimigos* (*T1*) e *Usar uma estratégia* (*T2*), respectivamente. Se houve falha nessas atividades, então é necessário que o jogador volte ao início da *quest*. O atributo *falha3* representa a falha da atividade *Capturar o Químico com vida* (*T6*). Falhar nesta atividade implica na falha de toda a missão. Por outro lado, o atributo *fim*, quando verdadeiro, representa a execução da missão com sucesso.

Para simular um possível cenário de como o MLM do jogo exemplificado funciona, é preciso então considerar as seguintes restrições:

- *T0* precisa ser executada por todos os jogadores;
- *T1* e *T2* representam desafios que os jogadores podem escolher executar sozinhos ou em grupo;
- *T5*, *T6*, *T7* e *T8* só podem ser executadas uma única vez, ou seja, uma vez executadas por um jogador os outros jogadores não poderão mais executá-las;
- *T8* representa o final da *quest* e deve ser executada por todos os jogadores.

Uma vez definida a classe de objetos, os atributos e as variáveis formais, é preciso definir então as funções de interpretação. A função de interpretação para cada transição do modelo é dada de acordo com a seguinte distribuição:

[1] Um tipo de dado primitivo que possui dois valores: verdadeiro ou falso.

$$\eta_x(T0) = \begin{cases} \text{verdadeiro } if(x.\neg fim) \\ \text{incerto } if(x.fim) \end{cases} \quad (1)$$

$$\eta_x(T1) = \begin{cases} \text{verdadeiro } if(x.\neg fim) \\ \text{incerto } if(x.fim) \end{cases} \quad (2)$$

$$\eta_x(T2) = \begin{cases} \text{verdadeiro } if(x.\neg fim) \\ \text{incerto } if(x.fim) \end{cases} \quad (3)$$

$$\eta_x(T3) = \begin{cases} \text{verdadeiro } if(x.\neg falha1) \\ \text{incerto } if(x.fim) \end{cases} \quad (4)$$

$$\eta_x(T4) = \begin{cases} \text{verdadeiro } if(x.\neg falha2) \\ \text{incerto } if(x.fim) \end{cases} \quad (5)$$

$$\eta_x(T5) = \begin{cases} \text{verdadeiro } if(x.\neg fim) \\ \text{incerto } if(x.fim) \end{cases} \quad (6)$$

$$\eta_x(T6) = \begin{cases} \text{verdadeiro } if(x.\neg falha3) \\ \text{incerto } if(x.fim) \end{cases} \quad (7)$$

$$\eta_x(T7) = \begin{cases} \text{verdadeiro } if(x.\neg fim) \\ \text{incerto } if(x.fim) \end{cases} \quad (8)$$

$$\eta_x(T8) = \begin{cases} \text{verdadeiro } if(x.fim \wedge y.fim) \end{cases} \quad (9)$$

$$\eta_x(T9) = \begin{cases} \text{verdadeiro } if(x.falha3) \\ \text{incerto } if(x.fim) \end{cases} \quad (10)$$

$$\eta_x(T10) = \begin{cases} \text{verdadeiro } if(x.falha1) \\ \text{incerto } if(x.fim) \end{cases} \quad (11)$$

$$\eta_x(T11) = \begin{cases} \text{verdadeiro } if(x.falha2) \\ \text{incerto } if(x.fim) \end{cases} \quad (12)$$

Para fins de simulação, considere o cenário em que os jogadores *J1* e *J2* já tenham executado a atividade *T0* com sucesso. Depois disso, cada jogador executará uma atividade diferente: *J1* executará *T1* e *J2* executará *T2* (situação ilustrada na figura 38). Em ambas atividades, os jogadores podem terminar com sucesso ou não. Considere que o jogador *J1* não obteve sucesso ao executar *T0*. Assim, após o disparo de *T1*, o atributo *falha1* será verdadeiro. *T10* é então disparada e o jogador *J1* retorna ao início da *quest* (lugar *P0*). Enquanto isso, *J2* executa a atividade *T2* com sucesso e segue adiante para as demais atividades. Enquanto *J1* tenta passar pela atividade *T1*, *J2* executa as atividades *T5*, *T6* e *T7* com sucesso. Quando *T7* é disparada, o atributo *fim* se torna verdadeiro, indicando que a *quest* foi completada com sucesso, apesar de *J1* não ter realizado todas as atividades.

O comportamento ideal é que os jogadores executem as atividades em grupo. Quando um jogador termina a *quest* sozinho, mas ainda existem jogadores envolvidos em outras

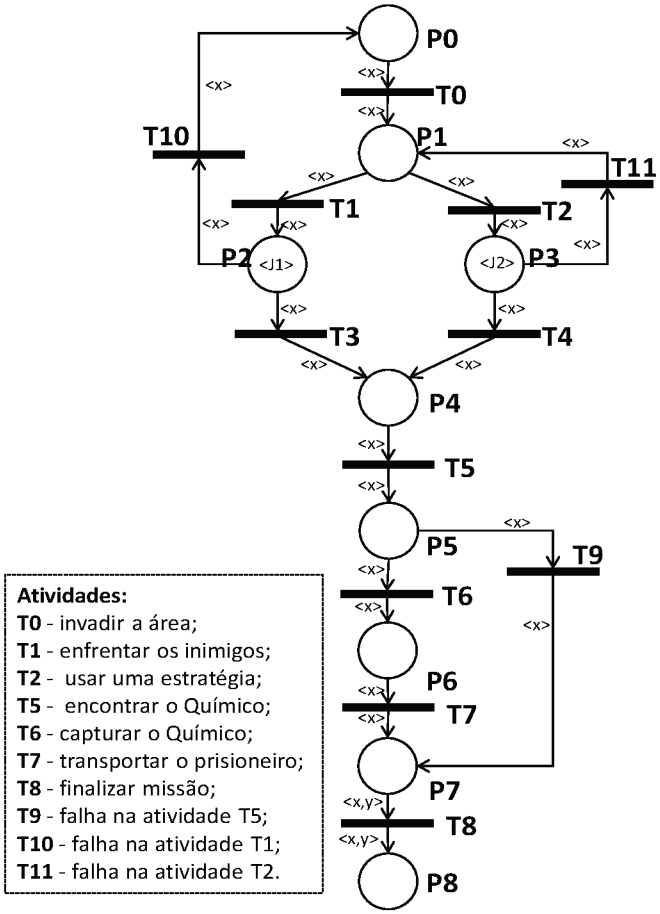


Figura 38 – Modelo Lógico Multiplayer da *quest The Chemist* após o disparo da transição *T1*, por *J1*, e *T2*, por *J2*.

atividades da mesma *quest*, do ponto de vista de uma representação formal das atividades, uma inconsistência pode ocorrer. Isso acontecerá em particular quando um dos jogadores conseguir concluir as atividades da missão; como consequência direta, os outros jogadores terão que cancelar a execução de algumas atividades que o outro jogador já executou. No caso específico do exemplo apresentado, as atividades são *T3*, *T5*, *T6* e *T7*. Uma maneira de resolver esse problema é usando o conceito de cancelamento de atividades apresentado em (REZENDE.; JULIA.; CARDOSO., 2016). Nesse trabalho, os autores abordaram o cancelamento de algumas atividades de um processo de negócio usando o conceito de pseudo-disparo de transição de uma WF-net possibilística. De acordo com Rezende., Julia. e Cardoso. (2016), as noções de cancelamento de atividade podem ser generalizadas para a noção de cancelamento de região, ou seja, uma região arbitrária do modelo pode ser submetida a uma ação de cancelamento.

Considere que o jogador *J2* executou todas as atividades com sucesso e suponha que o jogador *J1* se encontra no lugar *P2*, conforme ilustra a figura 39. Considerando uma rede de Petri clássica, o jogador *J1* deveria executar todas as atividades *T3*, *T5*, *T6* e *T7* para alcançar o final da *quest*. Mas devido ao fato do jogador *J2* já ter finalizado tais atividades, o objetivo principal de *J1* é simplesmente de alcançar o jogador *J2* no final da

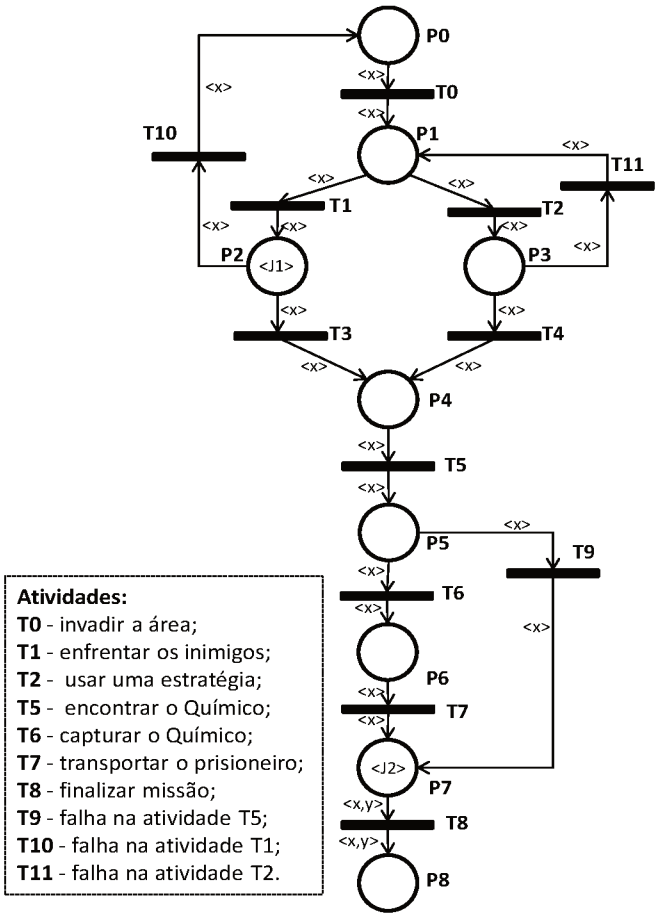


Figura 39 – Modelo Lógico Multiplayer da *quest The Chemist* após o disparo da transição *T0*, *T2*, *T4*, *T5*, *T6* e *T7* por *J1*, e *T0* e *T1*, por *J2*.

quest para que juntos, finalizem a missão (disparo da transição *T8*) e iniciam o próximo nível do jogo. Assim, o uso de pseudo-disparo, permite manter correta a semântica do jogo. A figura 40 ilustra a execução do cenário considerado que acontece da seguinte forma:

- ❏ se fim = verdadeiro, então η_x (T3) = incerto e T3 é pseudo disparada para o objeto J1;
- ❏ se fim = verdadeiro, então η_x (T5) = incerto e T5 é pseudo disparada para o objeto J1;
- ❏ se fim = verdadeiro, então η_x (T6) = incerto e T6 é pseudo disparada para o objeto J1;
- ❏ se fim = verdadeiro, então η_x (T7) = incerto e T7 é pseudo disparada para o objeto J1;
- ❏ se fim = verdadeiro e J1 e J2 estão no lugar P7, então η_x (T8) = verdadeiro. T8 é então habilitada por uma marcação incerta com uma interpretação verdadeira.

Consequentemente o algoritmo de recuperação, apresentado no capítulo 2, seção 2.3, é chamado para voltar à determinada marcação correspondente à figura 41.

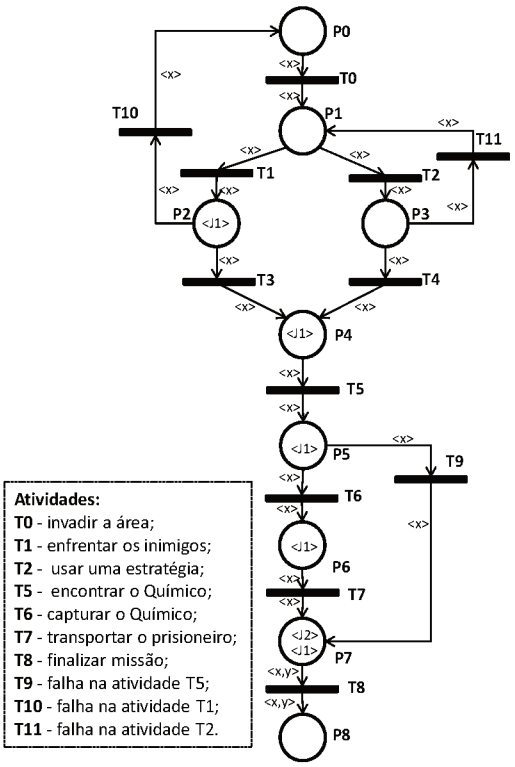


Figura 40 – Reresentação dos pseudos-disparos executados por J1.

O MLM da *quest The Chemist* é então finalizado apenas quando todos os jogadores estiverem no lugar *P7* do modelo. Se todas as atividades da *quest* fossem executadas por todos os jogadores desde o início da *quest*, então todas as transições do modelo seriam disparadas com certeza. Entretanto, uma vez que cada jogador tem autonomia para fazer o que desejar dentro do ambiente do jogo, mesmo quando o jogo tem como objetivo a cooperação entre as atividades, comportamentos possibilísticos são gerados devido as decisões individuais de cada jogador e precisam se expressar através do jogo de fichas do modelo (*Token Player*).

5.1.2 Modelo Topológico Multiplayer

Assim como no caso singleplayer, em um jogo multiplayer o mapa topológico é descrito como um conjunto de áreas significativas onde os jogadores executarão as atividades. As áreas são fixas durante todo o jogo e possuem ligações com outras áreas adjacentes. Algumas áreas do jogo podem ter, inicialmente, acesso livre ou possuir algum tipo de restrição. Nesse caso, uma atividade precisa ser executada para que o jogador consiga o acesso para a área em questão. Sendo assim, é possível descrever o mapa topológico de um jogo multiplayer em termos de um grafo de estados de acordo com a Definição 11 apresentada no capítulo 4.

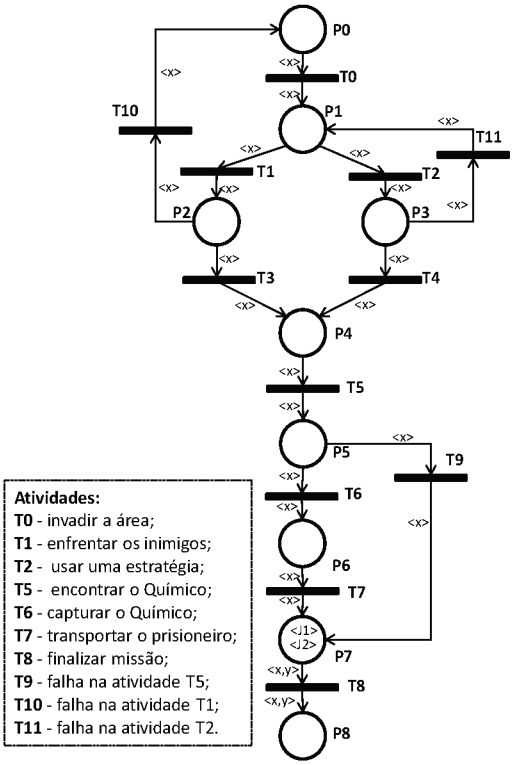


Figura 41 – Após o cancelamento dos pseudos-disparos.

Considere o exemplo do jogo *Tom Clancy's: Ghost Recon Wildlands* apresentado na seção anterior. Para executar as atividades da *quest The Chemist* os jogadores precisam estar em uma região específica do mapa de jogo, denominada *Marcavi*, apresentada na figura 42(a). As áreas da região onde serão executadas as atividades significativas do jogo aparecem enumeradas na figura 42(b).

(a) Mapa da região *Marcavi*.

(b) Áreas significativas para a *quest The Chemist*.

Figura 42 – *Marcavi*: mapa para a *quest The Chemist*.

A figura 43 ilustra o mapa topológico da *quest The Chemist* representada por um grafo de estados. A construção do mapa topológico por meio do grafo de estados tem o conjunto de áreas significativas do mapa dado por P = {1,2,3,4,5,6,7,8}. Já o conjunto

de fronteiras entre as áreas adjacentes é dado por T = {T1,T2,T3,T4,T5,T6,...,T18}.

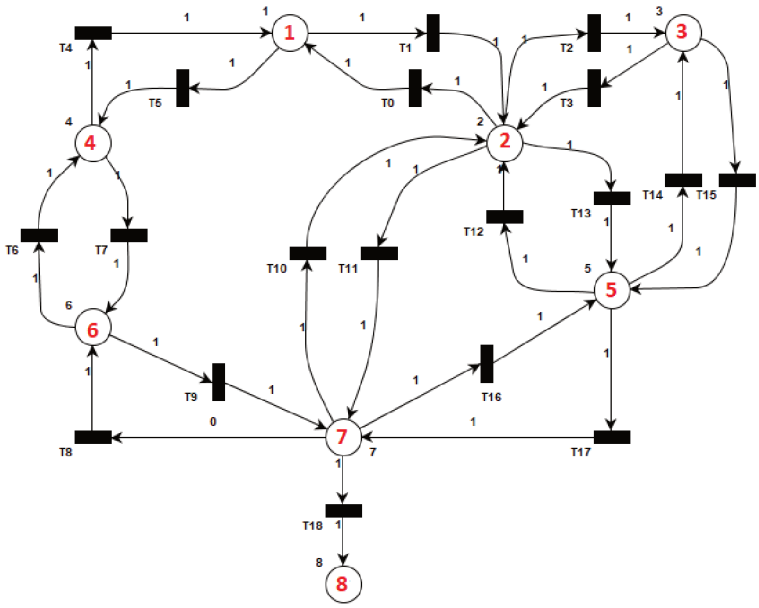


Figura 43 – Mapa topológico representado por um grafo de estados para a *quest The Chemist*.

A ficha em um lugar no grafo de estados representa a localização do jogador. Os disparos das transições representam então a locomoção do jogador de uma área para outra. De acordo com o grafo de estados apresentado na figura 43, as relações I e O que definem, para cada transição/fronteira, as áreas de entradas e saída são apresentadas a seguir:

- I(T0) = {2}, O(T0) = {1}
- I(T1) = {1}, O(T1) = {2}
- I(T2) = {2}, O(T2) = {3}
- I(T3) = {3}, O(T3) = {2}
- I(T4) = {4}, O(T4) = {1}
- I(T5) = {1}, O(T5) = {4}
- I(T6) = {6}, O(T6) = {4}
- I(T7) = {4}, O(T7) = {6}
- I(T8) = {7}, O(T8) = {6}
- I(T9) = {6}, O(T9) = {7}
- I(T10) = {7}, O(T10) = {2}

- I(T11) = {2}, O(T11) = {7}
- I(T12) = {5}, O(T12) = {2}
- I(T13) = {2}, O(T13) = {5}
- I(T14) = {5}, O(T14) = {3}
- I(T15) = {3}, O(T15) = {5}
- I(T16) = {7}, O(T16) = {5}
- I(T17) = {5}, O(T17) = {7}
- I(T18) = {7}, O(T18) = {8}

No Modelo Topológico Multiplayer, as interações entre os jogadores não geram incertezas a respeito da localização física do jogador no mapa, por se tratar de uma representação topológica do mundo do jogo. Portanto, no mapa multiplayer todos os disparos de transições serão sempre disparos certos.

5.2 Modelo de Simulação

A criação dos modelos de cenários de jogos multiplayer no software CPN Tools permite explorar, além da representação gráfica, as vantagens de implementação de tipos e estrutura de dados, funções, variáveis e funcionalidades de análise e simulação. A implementação dos modelos apresentados na seção 5.1 será realizada usando o CPN Tools. Portanto, os modelos serão adaptados para as Redes de Petri Coloridas.

5.2.1 Modelo Lógico Multiplayer Temporizado

Para a criação do Modelo Lógico Possibilístico no caso *multiplayer* no CPN Tools, é preciso considerar a criação de tipos de dados específicos, como o tipo de dado que representará o objeto *Jogador*. Um objeto deve possuir atributos que o caracterizam. No caso do tipo *Jogador*, esses atributos podem representar o identificador (nome ou apelido), a presença de um item, o status de uma atividade executada, o nível de habilidade, o limite de vida do personagem, entre outros aspectos. A definição de tais atributos dependerá dos objetivos do jogo, bem como das atividades a serem executadas.

Para exemplificar a implementação de um objeto do tipo *Jogador* e de seus atributos, considere novamente o exemplo do jogo *Tom Clancy's Ghost Recon: Wildlands*. Nesse exemplo, as informações necessárias do objeto para a execução das atividades da *quest* são apenas os status das atividades e o atributo identificador do jogador. Para representar o status da execução das atividades, pode-se usar um ou mais atributos associados ao

jogador e atualizá-los a cada execução de atividade. No exemplo aqui apresentado apenas dois atributos são necessários para registrar o status das atividades. O primeiro registrará se a atividade foi ou não realizada com sucesso, e o segundo registrará se a última atividade da *quest* foi executada. O modelo de simulação da *quest The Chemist* pode ser visto na figura 44.

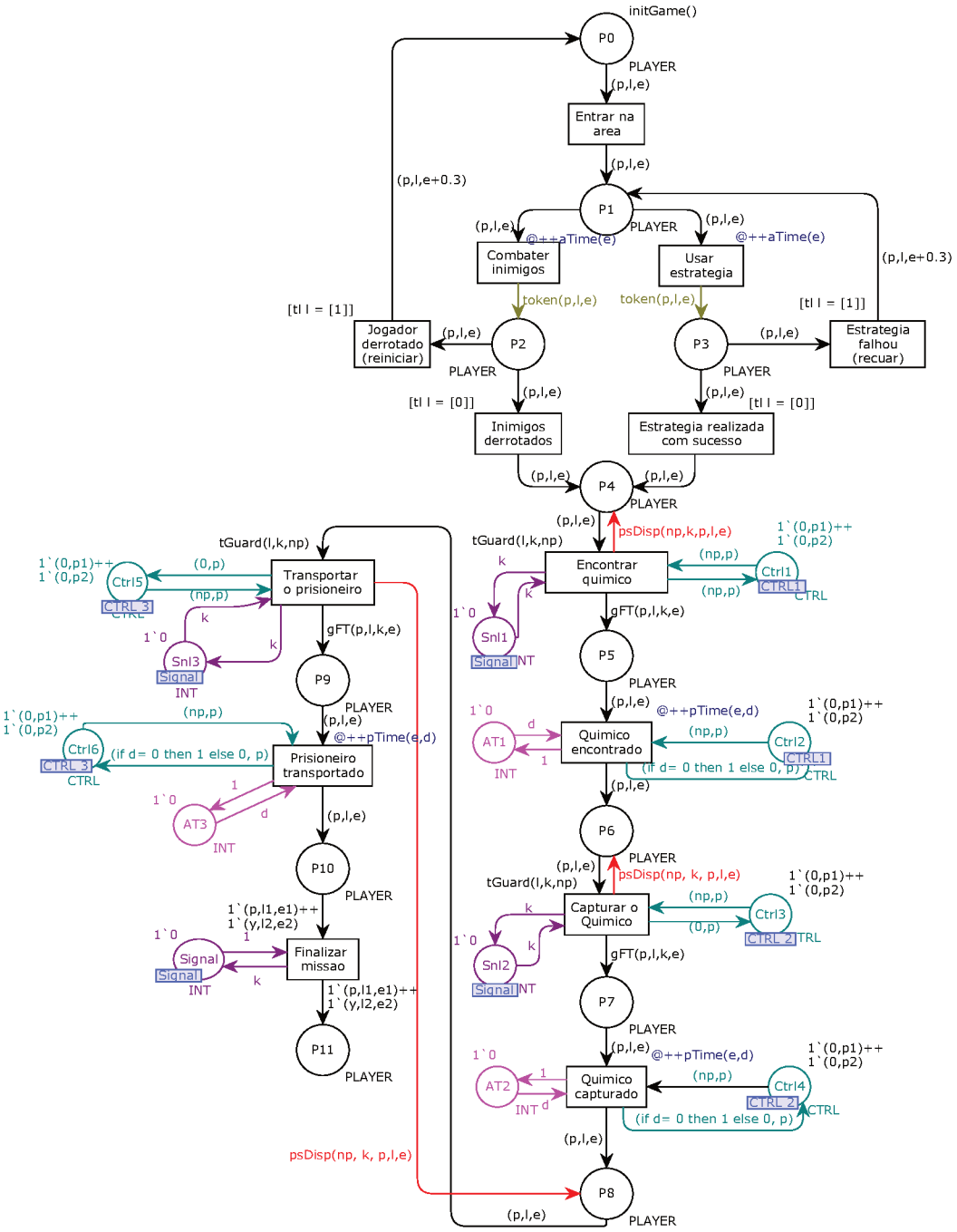


Figura 44 – Modelo Lógico Multiplayer Temporizado da *quest The Chemist* implementado no CPN Tools.

Na modelo ilustrado pela figura 44, o lugar *P0* representa o início da *quest* enquanto que o lugar *P11* representa o final. As transições correspondem as atividades. Todos

os lugares da rede pertencem à classe *Jogador*. No CPN Tools, a implementação da classe do objeto Jogador será realizada por meio da criação de um *color set*. O *color set PLAYER* representará um tipo de dado abstrato composto pelo identificador do jogador e os atributos de status. A implementação do *color set PLAYER* é dada a seguir.

```
1 colset P = with p1 | p2; (* tipo de dado que representa o identificador
     do jogador *)
2 colset L = list INT with 2..2; (* lista de atributos para representar os
     status das atividades do jogo *)
3 colset REAL = real; (*tipo de dado primitivo para representar o tempo de
     jogo*)
4 colset PLAYER = product P * L * REAL; (* tipo de dado que representa o
     objeto ‘‘Jogador’’ *)
```

O *color set P* define os identificadores dos jogadores podendo assumir valores de *p1* até p_n, onde n representa o número de jogadores. No caso do exemplo aqui apresentado, o valor de n é 2. O *color set L* define a lista de atributos do jogador, que neste exemplo corresponde aos status das atividades. Tais atributos podem assumir os valores do tipo inteiro que correspondem a *verdadeiro* (1), *falso* (0) e *incerto* (2). O *color set REAL* representa um tipo de dado cujo os valores pertencem ao conjunto dos números reais. *PLAYER* é então o *color set* que representa a classe do objeto *Jogador* formado pelo produto dos conjuntos *P*, *L* e *REAL*.

As variáveis do modelo têm papel fundamental no gerenciamento dos disparos, pois é por meio delas que os atributos serão atualizados a cada disparo de transição. Para a *quest The Chemist*, a definição das principais variáveis são apresentadas abaixo.

```
1 var p,y: P; (* variaveis do tipo de dado P *)
2 var l, l1, l2: L; (* variaveis do tipo lista *)
3 var np, k, d: INT; (* variaveis do tipo inteiro *)
4 var e, e1, e2: REAL; (*variaveis do tipo real*)
```

As variáveis *p* e *y* são para os dados do tipo *P*, ou seja, o identificador do jogador. As variáveis *np*, *k* e *d* podem assumir valores do tipo inteiro e servirão para gerenciar os disparos das transições atividades. As variáveis *l*, *l1*, *l2* pertencem ao tipo *L* e servirão para armazenar a lista de status das atividades. Por fim, as variáveis *e*, *e1* e *e2* são do tipo real e serão utilizadas para representar o nível de experiência do jogador. É por meio do nível de experiência que o cálculo do tempo de execução de cada atividade será feito. Todas as variáveis são adicionadas aos arcos do modelos e só poderão receber valores de mesmo tipo.

Lugares especiais precisam ser adicionados à estrutura do modelo para que as variáveis sejam atualizadas a cada disparo de transição. Os lugares com o prefixo *CTRL* atualizam a variável *np*, do tipo inteiro, que é responsável por permitir a execução e o cancelamento das atividades associadas aos pseudo-disparos. Os lugares *CTRL* são inicializados com o identificador do jogador e o valor de *np* (por exemplo, *1‘(p1,0)*), e são associados às

transições de início e fim de cada atividade que pertence à região de incerteza do modelo, como ilustra a figura 45. Os lugares com prefixo AT atualizam a variável d e são associados apenas as transições de fim de cada atividade da região de incerteza. A variável d tem como objetivo indicar quando a transição é disparada com certeza, ou seja, quando a atividade é finalizada. Nas regiões de incerteza também são adicionados lugares com o prefixo *Snl*, que atualizarão a variável k, responsável por indicar o status da última atividade da *quest*. Tais lugares são definidos como *fusion places* e fazem parte do mesmo conjunto de fusão, fazendo com que a variável k funcione como um tipo de variável global para o modelo.

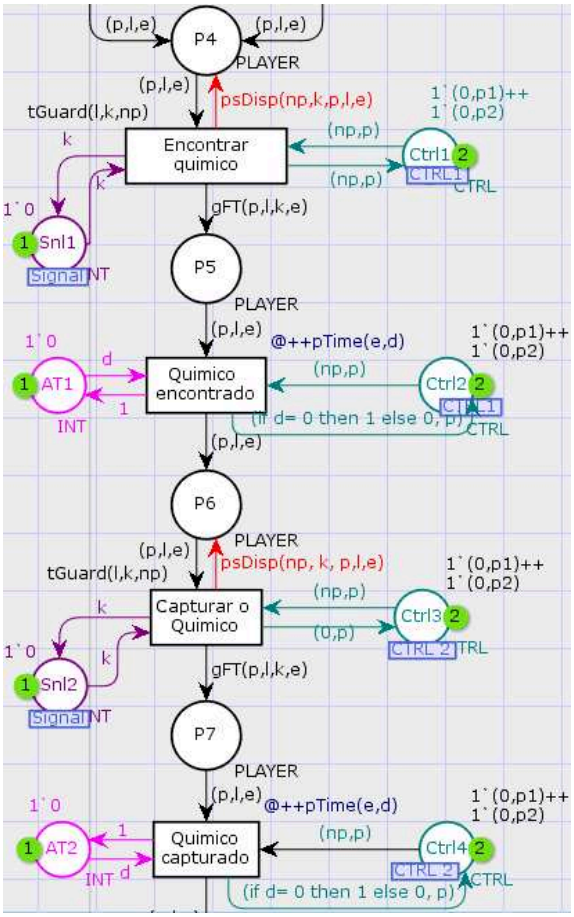


Figura 45 – Região de incerteza das atividades *Encontrar o Químico* e *Capturar o Químico* da *quest The Chemist*.

Para o funcionamento adequado do Modelo Lógico Multiplayer Temporizado é necessário a implementação de algumas funções. Na figura 44, a primeira função que aparece no modelo é a de distribuição de fichas *initGame()*, cuja implementação é apresentada a seguir.

```
1 (* Gera as fichas iniciais *)
2     fun initGame() = 1‘(p1, [0,0],0.5)++
3                      1‘(p2,[0,0],0.5);
```

A função *initGame()* gera duas fichas, pois no exemplo usado são apenas dois jogadores. No entanto, caso o número de jogadores seja maior do que 2, isso pode ser feito adicionando mais inscrições de ficha na função. A inscrição *1‘(p1,[0,0],0.5)* representa 1 jogador com identificador *p1*, com a lista de atributos *[0,0]* e com o nível de experiência inicial em *0.5*[2]. A figura 46 ilustra a distribuição de fichas da *quest The Chemist*.

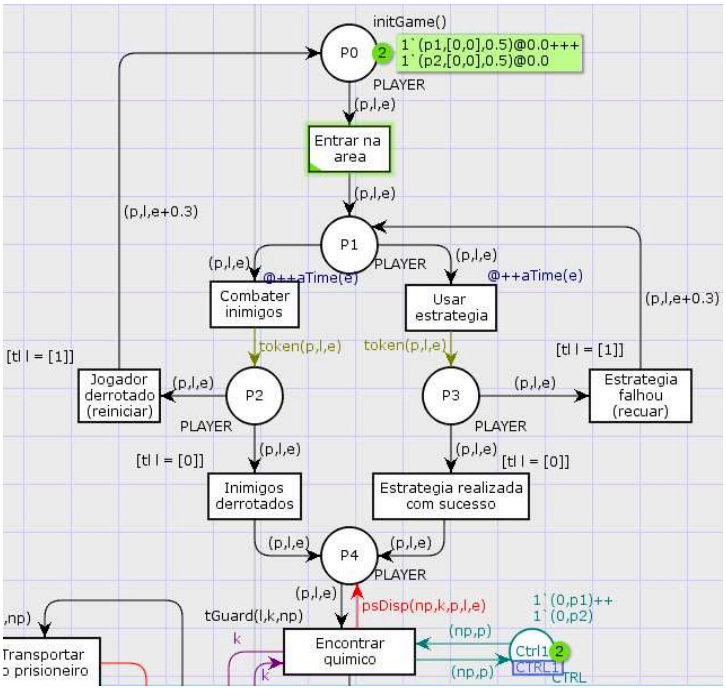


Figura 46 – Distribuição da marcação inicial no Modelo Lógico Multiplayer Temporizado.

Um aspecto que está presente na maioria dos jogos multiplayer é a possibilidade de falha dos jogadores ao executarem alguma atividade específica. Geralmente quando isso acontece, o jogador que falhou retorna para um momento anterior do jogo, permitindo que a atividade seja executada novamente. Em termos de modelo, o aspecto de falha de uma atividade pode ser representado por meio de um roteiro iterativo. Considere novamente o modelo da *quest The Chemist*; as atividades que possuem chances de falha são *Combater inimigos* e *Usar estratégia*. Tais atividades formam um roteiro iterativo com as transições *Jogador derrotado (reiniciar)* e *Estratégia falhou (recuar)*.

```
1 (* retorna 0 para sucesso; retorna 1 para falha *)
2     fun token(p:P, l:L, e:REAL) = (p, [hd l, FAIL.ran()],e);
```

A função *token(p,l,e)* é usada para definir se uma atividade foi ou não executada com sucesso. A função retorna uma expressão de arco com alteração no segundo atributo da lista de atributos por meio da função nativa do CPN Tools que distribui valores aleatórios, *ran()*. O *color set FAIL* pertence ao tipo de dado inteiro limitado aos valores *1* e *0*, sendo

[2] O valor do nível de experiência do jogador é um parâmetro que pode ser definido de acordo com os critérios e regras de cada jogo.

assim, *FAIL.ran()* gera valores aleatórios entre *0* e *1*. A figura 47 ilustra as transições *Combater inimigos* e *Usar estratégia* habilitadas.

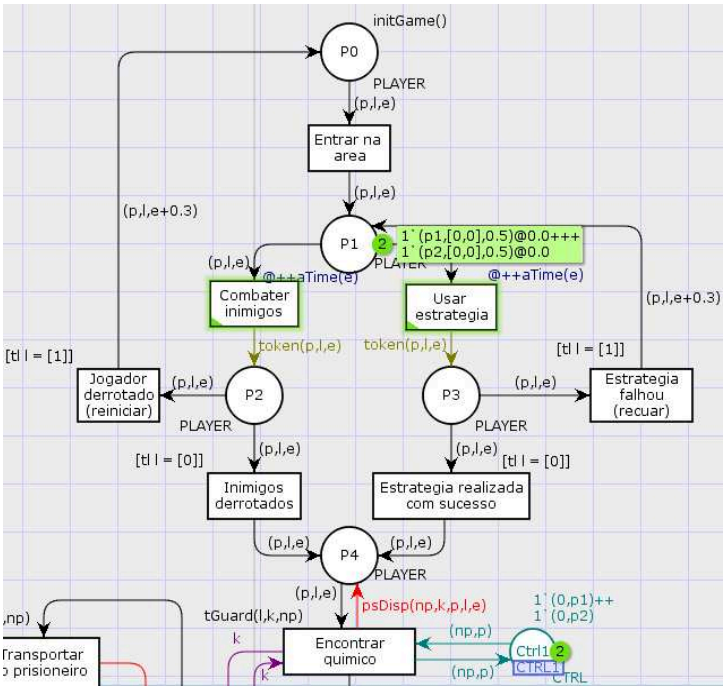


Figura 47 – Transições *Combater inimigos* e *Usar estratégia* habilitadas.

Se o jogador falhar na execução de uma dessas atividades, então o status da atividade receberá o valor *1* e o jogador deverá retornar ao ponto anterior para repeti-la, como ilustra a figura 48. Caso contrário, o status continua como está e o jogador segue adiante na *quest*. A verificação do status da atividade é feita por meio da função de guarda associada a transição correspondente. Por exemplo, na figura 48, a função *tl l = [0]*[3] é associada a transição *Inimigos derrotados* e verifica se o status da atividade é *1* (falhou) ou *0* (não falhou).

Outro aspecto importante que está presente nos jogos multiplayer é o compartilhamento das atividades. O compartilhamento pode ser cooperativo quando é necessário ter dois ou mais jogadores para executar uma atividade, ou exclusivo quando uma atividade só pode ser executada apenas uma única vez no jogo. A interação entre os jogadores faz com que atividades de compartilhamento exclusivo gerem incertezas no modelo. Para lidar com esse problema, o Modelo Lógico Multiplayer (MLM) baseado nas WorkFlow nets Possibilísiticas foi apresentado na seção 5.1. Uma forma de representar o modelo de simulação correspondente ao MLM é criar uma representação gráfica para a região de incerteza que permite determinar o início e fim das atividades compartilhadas, executar os disparos certos e incertos (pseudo-disparos) e aplicar a confirmação do disparo certo e/ou o cancelamento dos pseudo-disparos quando necessário.

[3] A expressão *tl l* e *hd l* são nativas do CPN Tools e servem para recuperar a cauda e a cabeça de uma lista, respectivamente

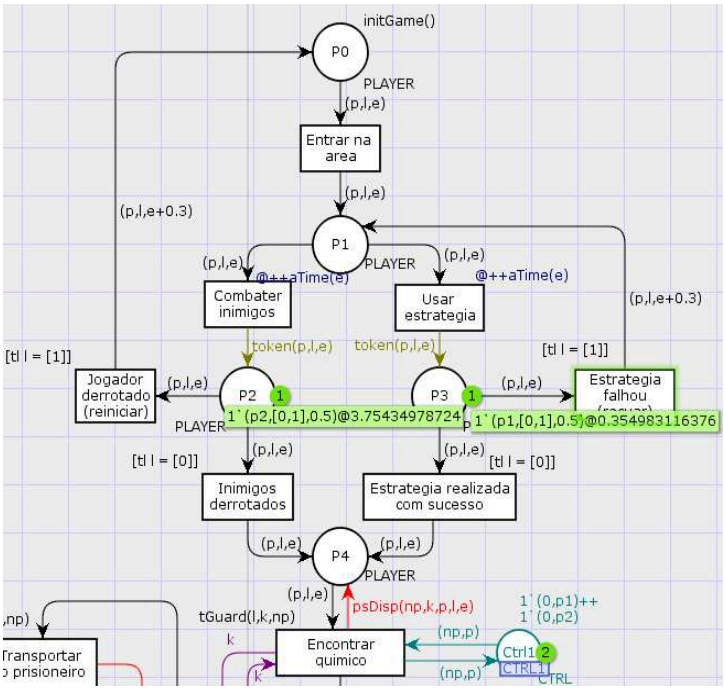
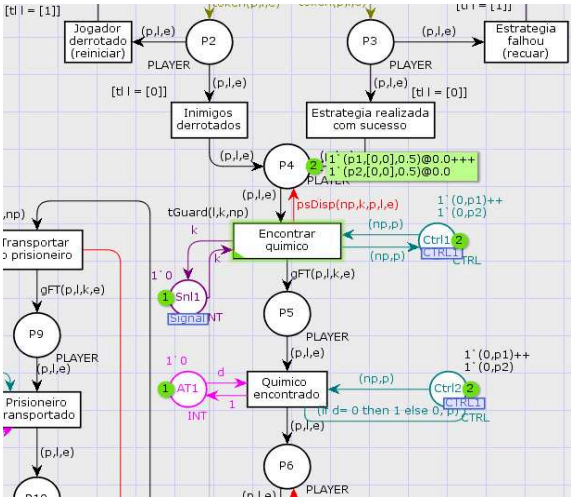


Figura 48 – Ao disparar a transição *Usar estratégia*, a função *token(p,l,e)* determinou que o jogador *P1* falhou ao executar a atividade, sendo assim, a transição *Estratégia fallhou (recuar)* é habilitada.

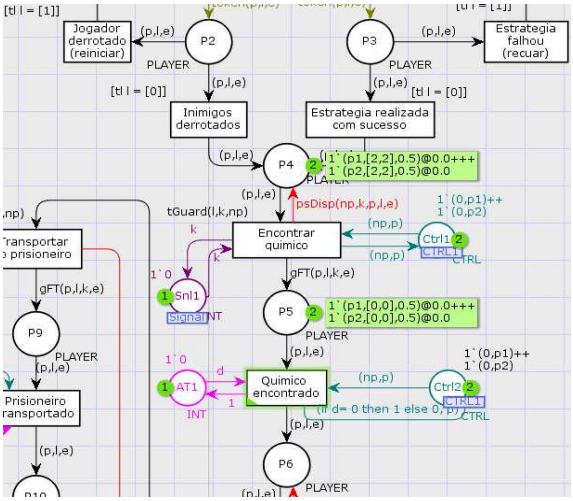
Considere o modelo da figura 44. Na *quest The Chemist* a atividade *Finalizar missão* corresponde ao caso de um compartilhamento cooperativo, pois é necessário que todos os jogadores a realizem juntos. Já as atividades *Encontrar o Químico*, *Capturar o Químico* e *Transportar o prisioneiro* correspondem a um compartilhamento exclusivo, ou seja, só podem ser executadas uma única vez. Para cada uma dessas atividades é necessário definir no modelo uma estrutura que especifique seu início e fim. Por exemplo, para a atividade *Encontrar o Químico* a transição *Encontrar químico* determina o início e a transição *Químico encontrado* determina o fim da atividade. Portanto, quando uma ficha se encontra no lugar entre as duas transições (lugar *P5*), significa que a atividade está sendo executada pelo jogador.

A estrutura de início e fim que demarca uma atividade compartilhada no jogo permite que os jogadores iniciem a atividade juntos. Ao iniciarem uma atividade compartilhada, o disparo da transição correspondente será incerto para todos os jogadores, pois não se sabe ainda quem terminará de fato a atividade. Observe o exemplo ilustrado na figura 49(a), onde os dois jogadores se encontram no lugar *P4*. A transição *Encontrar quimico* é então pseudo-disparada para ambos os jogadores, indicando que começaram a atividade juntos (figura 49(b)). O jogador *p1* é o primeiro a terminar, portanto, a transição *Quimico encontrado* é disparada pela primeira vez (figura 49(c)). Consequentemente, o disparo de *p1* é confirmado (figura 49(d)). Como a ação de encontrar o químico não pode ser feita mais de uma vez, o jogador *p2* poderá seguir adiante na *quest* sem finalizar a

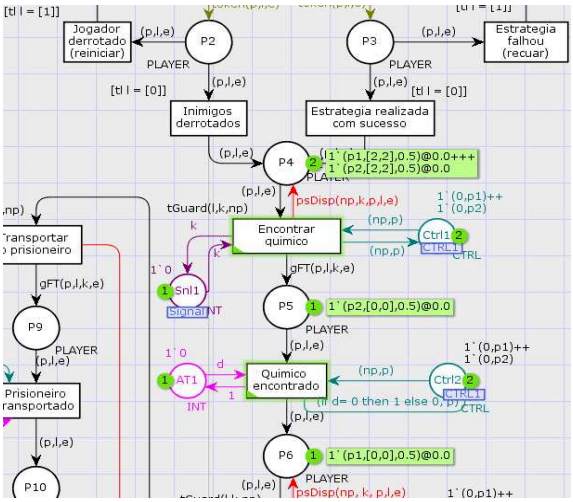
execução da atividade. Assim, o pseudo-disparo para *p2* é mantido, indicando que ele não finalizou a atividade. A função *tGuard(l,k,np)* associada as transições também faz parte do mecanismo para controlar os disparos, verificando por meio dos atributos se a ficha tem ou não permissão para disparar a transição. Por exemplo, uma ficha gerada por pseudo-disparo só poderá disparar uma determinada transição caso a confirmação ou o cancelamento do pseudo-disparo for acionado.



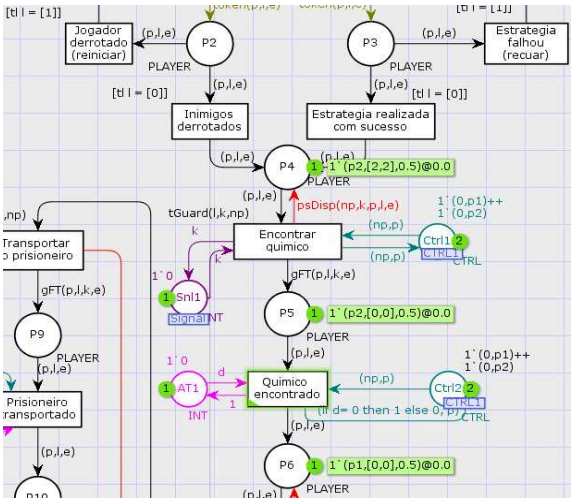
(a) Transição *Encontrar quimico* habilitada.



(b) Após o pseudo-disparo da transição *Encontrar quimico* para *p1* e *p2*.



(c) Após o primeiro disparo válido da transição *Quimico encontrado* por *p1*.



(d) Após a confirmação do disparo da transição *Encontrar quimico* por *p1*.

Figura 49 – Execução da atividade *Encontrar o Químico*.

As variáveis *np*, *k* e *d*, servem para controlar os disparos das transições no modelo, usando para isso a seguinte função com seus parâmetros *psDisp(np,k,p,l,e)*. Essa função é associada aos arcos de saída das transições que fazem parte do comportamento possibilístico, para os seus respectivos lugares precedentes. A função *psDisp(np,k,p,l,e)*, é definida como se segue.

```
1  fun psDisp(np:INT, k:INT, p:P, l:L, e:REAL) =
```

```
      if np = 0 andalso k = 0
      then 1`(p,[2, 2],e)
      else 0`(p,l,e);
```

A função *psDisp* recebe como parâmetro *np*, *k*, o identificador do jogador (p), a lista de atributos do jogador (l), e seu nível de experiência (e). Essa função é formada com a estrutura *IF THEN ELSE*, verificando se a atividade foi finalizada e se a *quest* foi encerrada. Se a condição for satisfeita, então um pseudo-disparo acontece. O pseudo-disparo deposita uma ficha no lugar de saída da transição (executado no modelo pela função *gFT(p,l,k,e)*, apresentada a seguir), mas não retira as instâncias do lugar de entrada. Assim, a função *psDisp* altera a lista de atributos da ficha para o valor 2, indicando que a ficha é gerada por um pseudo-disparo.

```
fun gFT(p:P, l:L, k:INT, e:REAL) =
if hd l = 2 andalso tl l = [2]
then 0`(p,l,e)
else 1`(p,l,e);
```

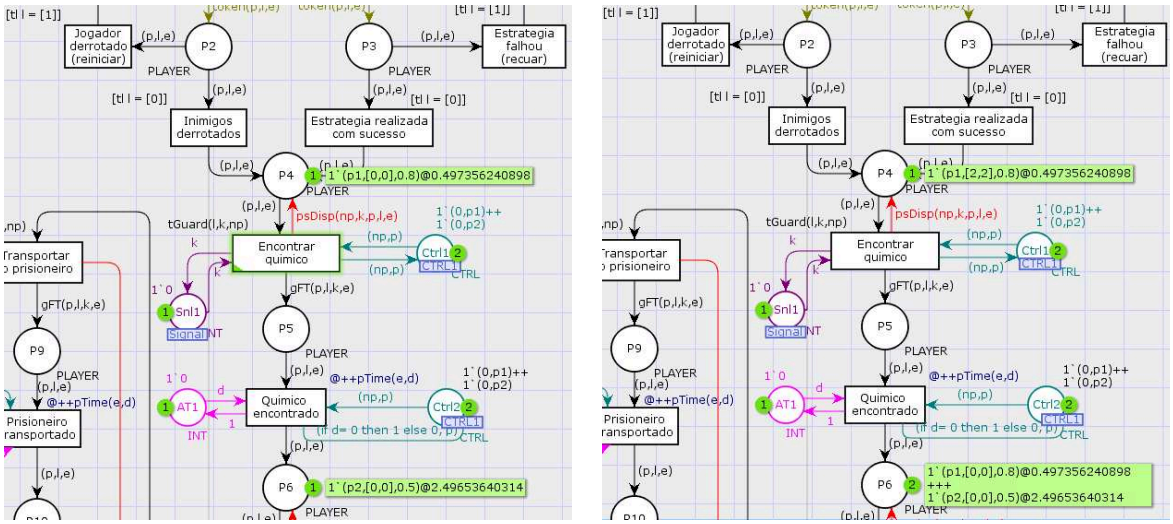
Quando se trata de um modelo para simulação de cenários de jogos é importante considerar o tempo estimado para a execução das atividades. Cada atividade do jogo possui um tempo de execução. Na abordagem proposta nesta pesquisa para o modelo de simulação de cenários multiplayer, o tempo de jogo será medido de acordo com o nível de experiência do jogador. Para tanto, a função exponencial será usada para gerar um tempo aleatório de execução para cada atividade. A função exponencial é uma das funções nativas do CPN Tools.

```
fun aTime(e:REAL) = exponential(e);

fun pTime(e:REAL,d:INT) =
  if d = 1
    then 0.0
  else exponential(e);
```

A função *aTime(e)* tem como parâmetro o nível de experiência do jogador e é associada as transições atividades do modelo. A função *pTime(e,d)* também gera o tempo de execução aleatório baseado na função exponencial, porém é associada as transições da região de incerteza que representam a finalização da atividade. No caso da transição atividade ser pseudo-disparada, o tempo de execução daquela atividade não é somado ao tempo de jogo do jogador. Essa situação pode ser exemplificada por meio da figura 50(a), onde o jogador *p2* já executou a atividade de encontrar o Químico. Então o jogador *p1* não precisará executar a atividade novamente, disparando as transições através de disparos incertos. O disparo incerto não contabiliza o tempo da atividade para *p1*, como ilustra a figura 50(b).

A utilização de funções, variáveis e tipos de dados específicos faz com que seja possível simular o funcionamento do Modelo Lógico Multiplayer Temporizado. O uso dos pseudo-



(a) Disparo da transição *Quimico encontrado* por p2.

(b) Pseudo-disparo da transição *Encontrar quimico* por p1.

Figura 50 – Execução da atividade *Encontrar o Químico* contabilizando o tempo.

disparos torna possível alterar os cenários do jogo (no caso *multiplayer*, os cenários de cada jogador são definidos somente em termos de possibilidade) sem mudar a sua estrutura. Os lugares especiais que são adicionados ao modelo trazem pouca complexidade e são eficientes para atualizar as variáveis locais e globais que são necessárias para efetuar os disparos das transições atividades sensibilizadas.

5.2.2 Modelo Topológico Multiplayer Temporizado

Neste trabalho, a proposta de implementação para o modelo de simulação do Mapa Topológico Multiplayer Temporizado considera as áreas significativas do mapa, bem como as fronteiras entre as áreas adjacentes. Os lugares da rede que representam as áreas do mapa serão do tipo *PLAYER*. Assim, cada lugar da rede poderá receber fichas correspondentes aos jogadores. A atribuição do tipo *PLAYER* aos lugares do mapa é importante quando se considera a possível interação do modelo do mapa topológico com o modelo lógico do jogo. Dessa forma, os modelos poderão se comunicar sem incompatibilidade de tipos de variáveis.

As fronteiras entre as áreas são expressas por meio das transições. Para simular o tempo de movimentação do jogador, será considerada a função de distribuição uniforme que produz um número aleatório entre um valor mínimo (a) e um valor máximo (b), com média $(a + b)/2$. Os valores máximo e mínimo podem ser alterados de acordo com as especificações do jogo. A função uniforme também é nativa do CPN Tools e no modelo proposto a sua implementação é dada como *mpTime()*.

```
1 fun mpTime() = uniform(a,b);
```

Considere novamente a *quest The Chemist*, apresentada na seção anterior. O modelo de simulação do mapa topológico multiplayer da *quest* pode ser visto na figura 51. A função *initGame()* usada inicializa as fichas da rede de acordo com o número de jogadores.

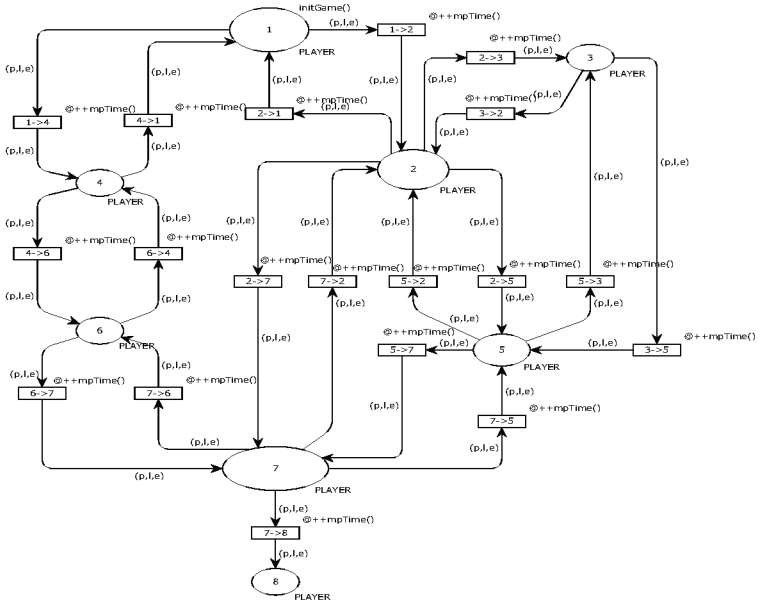
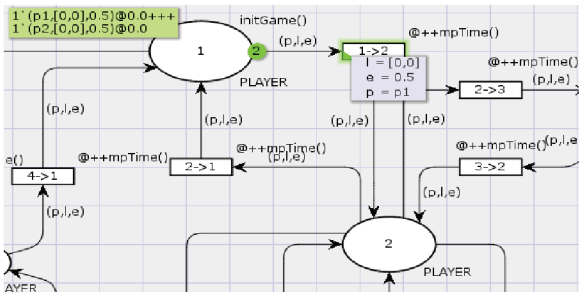
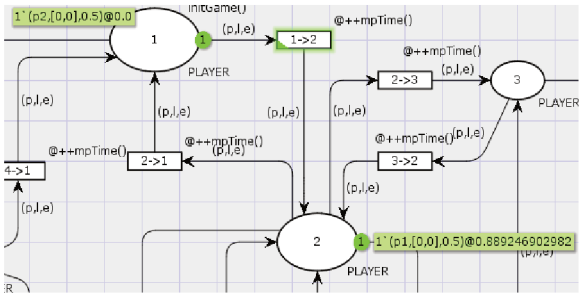


Figura 51 – Mapa Topológico Multiplayer Temporizado da *quest The Chemist*.

Associada às transições do modelo está a função *mpTime()* com os valores de $a = 0.1$ e $b = 1.0$. O disparo de uma transição representará então o tempo de movimentação do jogador para passar de uma área para a outra. Por exemplo, considere que o jogador *p1* se locomove da *área 2* para a *área 3* (figura 52(a)). De acordo com a função *mpTime()*, o jogador *p1* leva aproximadamente 0,88 unidades de tempo para ir da área 2 à área 3, como ilustra a figura 52(b).



(a) Estado inicial da rede.



(b) Estado da rede após o disparo da transição *1->2*.

Figura 52 – Mapa Topológico Multiplayer Temporizado da *quest The Chemist*.

5.2.3 Mecanismo de Comunicação

Para estabelecer a comunicação entre o Modelo Lógico Multiplayer e o Modelo Topológico Multiplayer, a fim de gerar um Modelo de Jogo Global Multiplayer, é preciso estabelecer um mecanismo para representar e gerenciar a execução das atividades em áreas específicas, a liberação de áreas do mapa de acordo com a finalização das atividades, e as interações entre os jogadores. Considere como exemplo um cenário de jogo multiplayer hipotético onde existe uma atividade *A1* para ser executada e duas áreas do mundo virtual, *M1* e *M2*. Para este exemplo, considere as seguintes regras:

1. a atividade *A1* só pode ser executada uma única vez;
2. a atividade *A1* é executada na área *M1*;
3. o jogador só poderá passar para a área *M2* após executar a atividade *A1*.

O exemplo citado acima é ilustrado na figura 53. O lado esquerdo da figura representa o Modelo Lógico Multiplayer, enquanto que o lado direito representa o Modelo Topológico Multiplayer. A primeira regra do jogo afirma que só um jogador poderá executar a atividade *A1*, ou seja, no modelo da figura 53, a transição atividade *A1* irá disparar com certeza para um jogador e para o outro fará um disparo incerto (pseudo-disparo). A segunda regra afirma que para executar *A1* é preciso que o jogador esteja localizado na área *M1*, ou seja, é preciso ter uma ficha do jogador no lugar *M1* do Mapa Topológico Multiplayer. A terceira regra diz respeito a uma condição para a liberação da área *M2*, representada no modelo da figura 53 pelo lugar *Cond1*. A transição *F1*, que representa a fronteira entre *M1* e *M2*, só será disparada quando houver uma ficha no lugar *Cond1*, indicando que a condição é satisfeita, e uma ficha do tipo jogador no lugar *M1*. O lugar *RT* e a transição *RTf* fazem parte do mecanismo de comunicação para gerenciar a dinâmica dos disparos incertos, quando necessário. *J1* e *J2* são os objetos que pertencem à classe Jogador.

É importante ressaltar que o mecanismo de comunicação aqui apresentado é baseado em uma rede de Petri Colorida. Esse tipo de rede permite gerenciar os disparos por meio de funções associadas as transições e/ou aos arcos de saída. No entanto, no exemplo da figura 53 os detalhes de implementação das funções foram omitidos, mas serão apresentados na próxima seção. Para simular o funcionamento do mecanismo de comunicação (baseado numa rede de Petri Colorida) no exemplo da figura 53, considere que o jogador *J1* irá executar a atividade *A1*, como ilustra a figura 54. Por ser o primeiro disparo, a interpretação da transição é verdadeira, então um disparo certo acontece. Após o disparo certo de *A1* uma ficha é produzida no lugar *Cond1*, no lugar *P2* e outra no lugar *M1* (figura 55). Uma vez que a condição de liberação da área *M2* é satisfeita, então os jogadores estão aptos para passar da área *M1* para *M2*, como ilustra a figura 56.

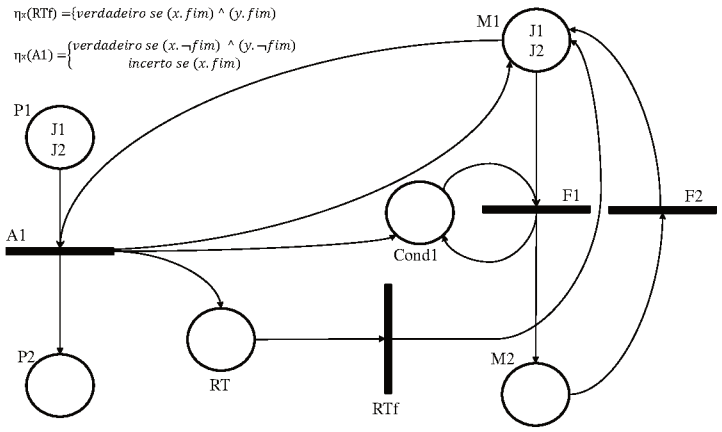


Figura 53 – Comunicação entre modelos lógico e topológico multiplayer.

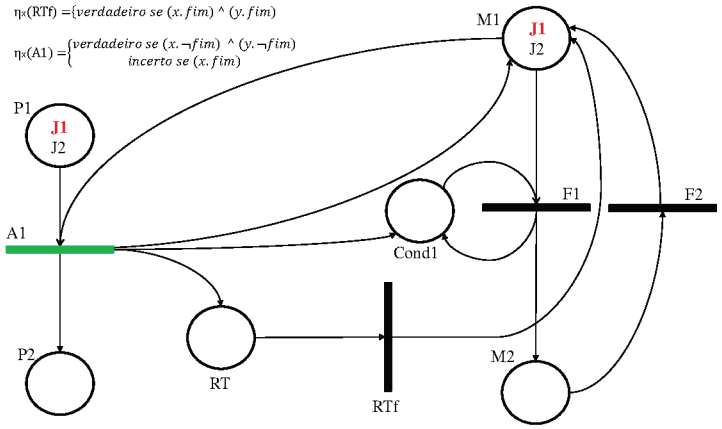


Figura 54 – Transição atividade *A1* habilitada por *J1*.

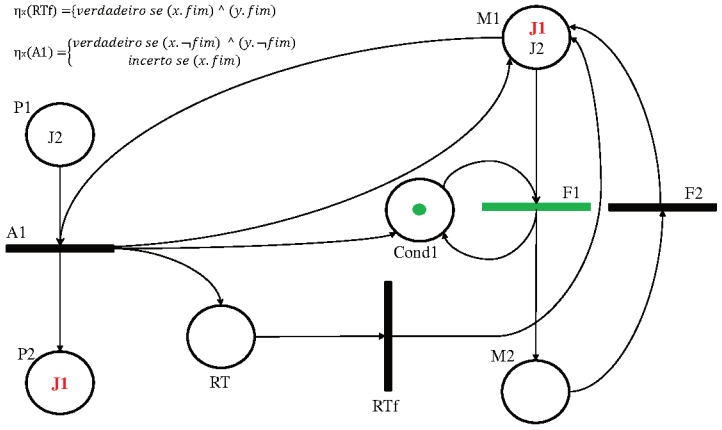


Figura 55 – Após o disparo de *A1* por *J1*.

Uma vez que *A1* já foi executada por um jogador, ela não precisará ser executada novamente para os outros jogadores. Portanto, a interpretação da transição atividade *A1* é incerta e será pseudo-dispara para os demais jogadores, como ilustrado na figura 57.

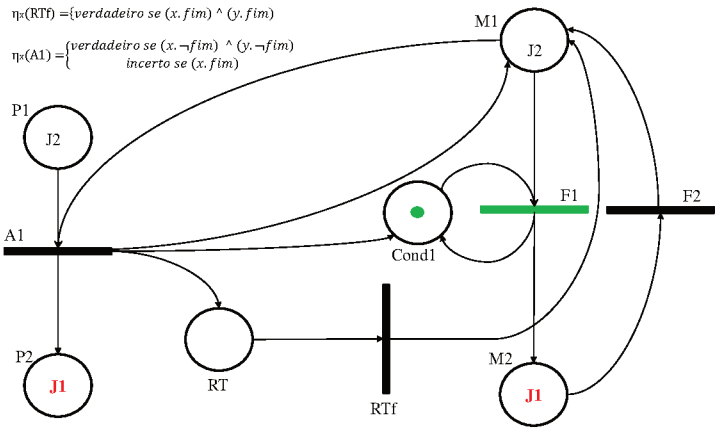


Figura 56 – Disparo da transição *F1* por *J1*.

Quando *A1* é pseudo-disparada por *J2* (figura 58), uma ficha é adicionada no lugar *RT*, no lugar *P2* e no lugar *M1*. A instância do lugar de entrada de *A1* no modelo lógico é mantida.

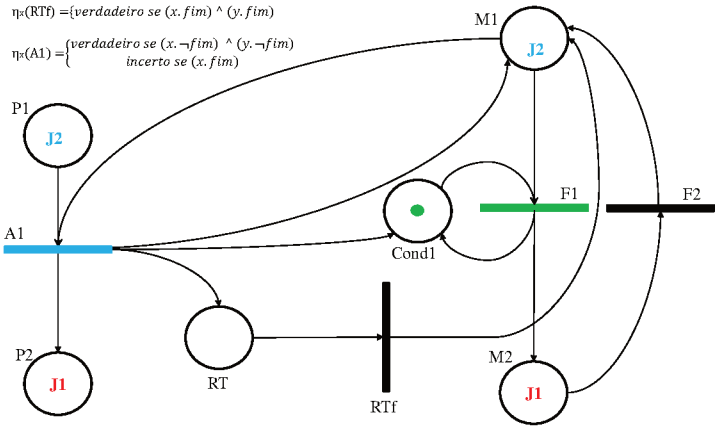


Figura 57 – Transição *A1* habilitada com interpretação incerta.

Após a execução de todas as atividades do jogo, a interpretação da transição *RTf* será verdadeira e então um disparo de *RTf* acontecerá (fig:exemploMC7). Assim, *RTf* será disparada, gerando uma ficha no lugar *M1*, como ilustra a figura 60.

O lugar *RT* no mecanismo de comunicação tem o objetivo de funcionar como um recurso para armazenar a ficha do jogador que executou o pseudo-disparo da transição. Armazenar essa informação é importante para a realização do cancelamento do pseudo-disparo ao final do nível de jogo. O lugar *RT* faz então a comunicação com o mapa topológico informando para qual jogador o cancelamento será aplicado. No cenário de jogo, o cancelamento do pseudo-disparo de uma transição não gera novas fichas no modelo lógico nem no modelo topológico, como ilustra a figura 61.

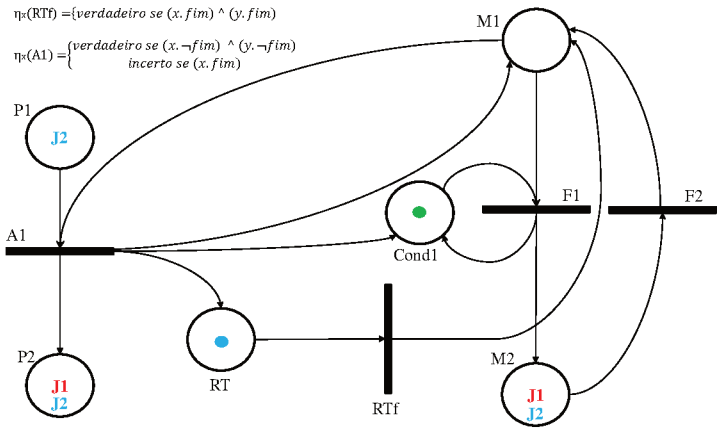


Figura 58 – Pseudo-disparo de *A1* por *J2*.

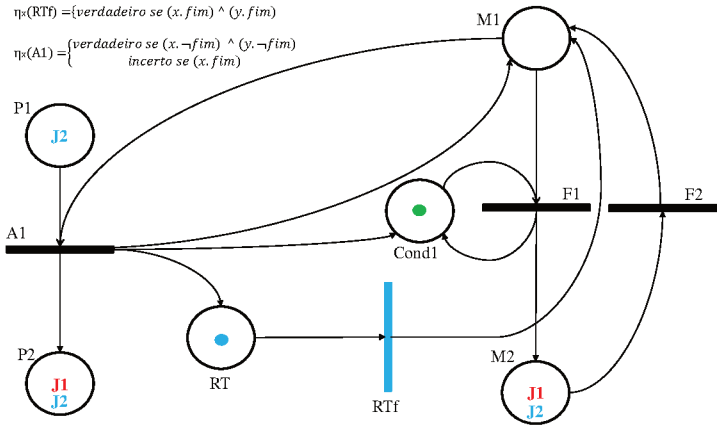


Figura 59 – *RTf* habilitada com interpretação verdadeira.

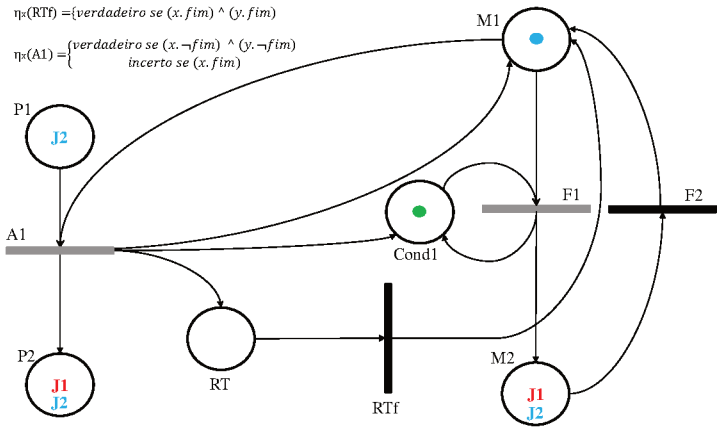


Figura 60 – Após o disparo da transição *RTf*.

5.2.4 Modelo Global Multiplayer Temporizado

O Modelo Global Multiplayer é formado pelo Modelo Lógico Multiplayer, Modelo Topológico Multiplayer e o mecanismo de comunicação. Uma vez que o modelo global

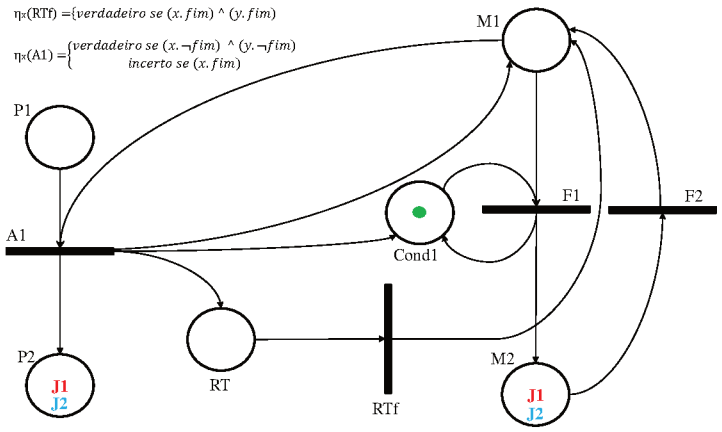


Figura 61 – Estado da rede após a execução do cancelamento do pseudo-disparo de *A1* por *J2*.

é estabelecido, o cenário de jogo pode ser simulado e analisado como um todo. Para explicar a proposta de implementação do modelo de simulação global no caso multiplayer será usado um exemplo simples de jogo com três atividades no modelo lógico e duas áreas no modelo topológico, como ilustra a figura 62. A utilização do CPN Tools, além de fornecer ferramentas para o gerenciamento do mecanismo de comunicação, ainda mantém os dois modelos visualmente distintos.

No modelo de simulação da figura 62, todos os lugares da rede pertencem à classe *PLAYER*. A classe *PLAYER* é composta por três atributos principais: a identificação do jogador (representado por *p1* e *p2*), uma lista de atributos de tamanho 2 que representa o status das atividades, e um indicador de experiência (*e*). Portanto, a ficha que representa o objeto jogador é dada por *1‘(p1,[0,0],0.5)*. O lado esquerdo da figura ilustra o Modelo Lógico Multiplayer e o lado direito ilustra o Modelo Topológico Multiplayer. As atividades *A1* e *A2* só podem ser executadas uma única vez, portanto, as transições *Inicio de A1*, *Fim de A1*, *Inicio de A2* e *Fim de A2* formam regiões de incerteza no modelo. Os lugares *M1*, *M2* e *M3* correspondem as áreas do mapa topológico.

A comunicação entre os modelos multiplayer acontecerá conforme o mecanismo apresentado na figura 53. Para a implementação do Modelo Global Multiplayer no CPN Tools, o conceito de *fusion places* é utilizado. Observe a figura 63 que destaca o mecanismo de comunicação implementado no CPN Tools. Os lugares do mapa topológico estão marcados com uma *fusion tag* correspondente à área. Por exemplo, os lugares *M1*, *m1* e *m11* estão rotulados com a *fusion tag M1*, pois todos representam a área do mapa *M1*. O mesmo acontece com *RT1* e *RTf1*, ambos pertencentes ao mesmo conjunto de fusão *RT1*. Os lugares que representam uma condição necessária para a liberação de uma área do mapa são *cond1* e *C1*, ambos pertencentes ao conjunto de fusão *C1*.

Para o gerenciamento dos pseudo-disparos no CPN Tools, duas funções são necessárias no mecanismo de comunicação: *rtDisp()* e *arcM()*. A função *rtDisp()* é associada ao arco

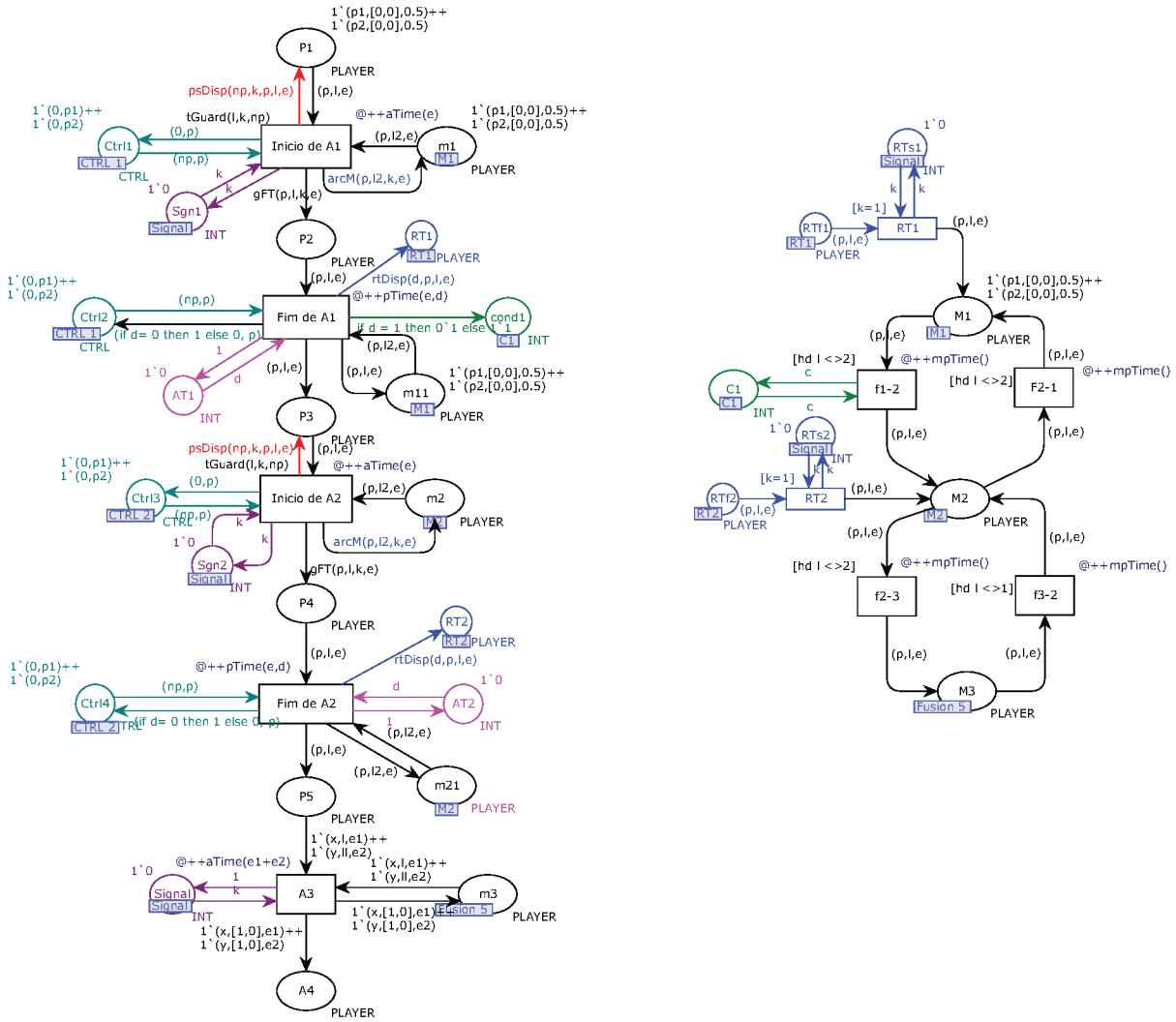


Figura 62 – Exemplo de um modelo de simulação global no caso multiplayer.

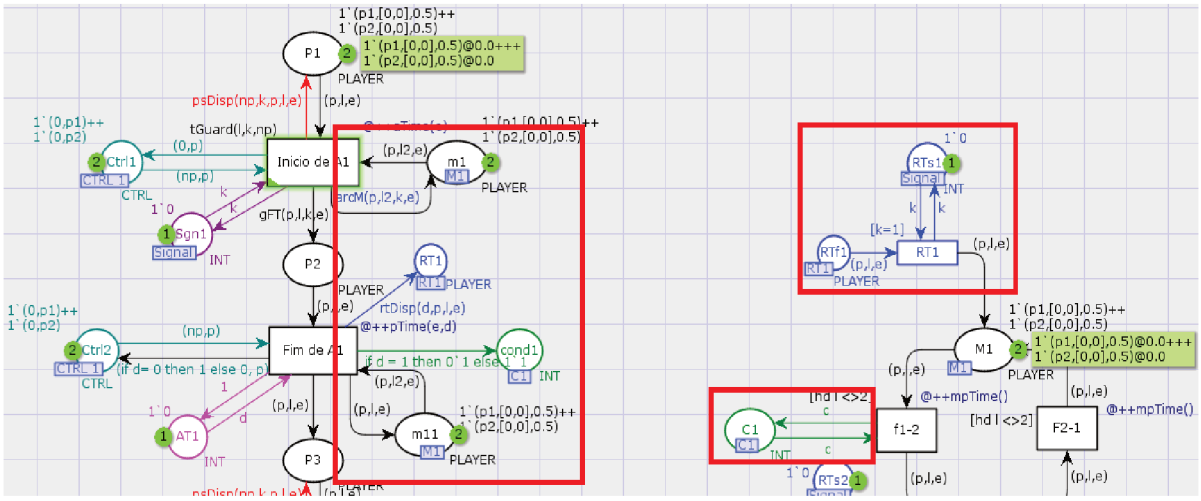


Figura 63 – Mecanismo de comunicação no Modelo Global Multiplayer.

de entrada dos lugares de prefixo *RT*, com o objetivo de armazenar no lugar a ficha do jogador que executou o pseudo-disparo. Já a função *arcM()* tem como objetivo de gerar fichas apenas enquanto a quest não for finalizada. A implementação dessas funções é dada como se segue.

```
fun rtDisp (d:INT, p:P, l:L,e:REAL) =
  if d=1 then          (*a transicao foi pseudo-disparada*)
    1‘(p,[2, 2],e)     (*gera a ficha que executou o pseudo-disparo*)
  else 0‘(p,l,e);      (*caso contrario, nao gera fichas*)

  fun arcM(p:P, l:L, k:INT) =
  if k = 0             (*a quest n o foi finalizada*)
    then 1‘(p,l)       (*gera a ficha do jogador normalmente*)
  else 0‘(p,l);        (*caso contrario, nao gera fichas*)
```

Para exemplificar o funcionamento do modelo global, considere o exemplo apresentado na figura 62. Inicialmente os jogadores se encontram no lugar *M1* e a primeira ação é executar a atividade *A1*. Pela marcação especificada no modelo de simulação, o jogador *p2* terminou primeiro a atividade *A1*, portanto o jogador *p1* pseudo-dispara a transição atividade *A1*, e uma ficha de *p1* é armazenada no lugar *RT1*, como ilustra a figura 64.

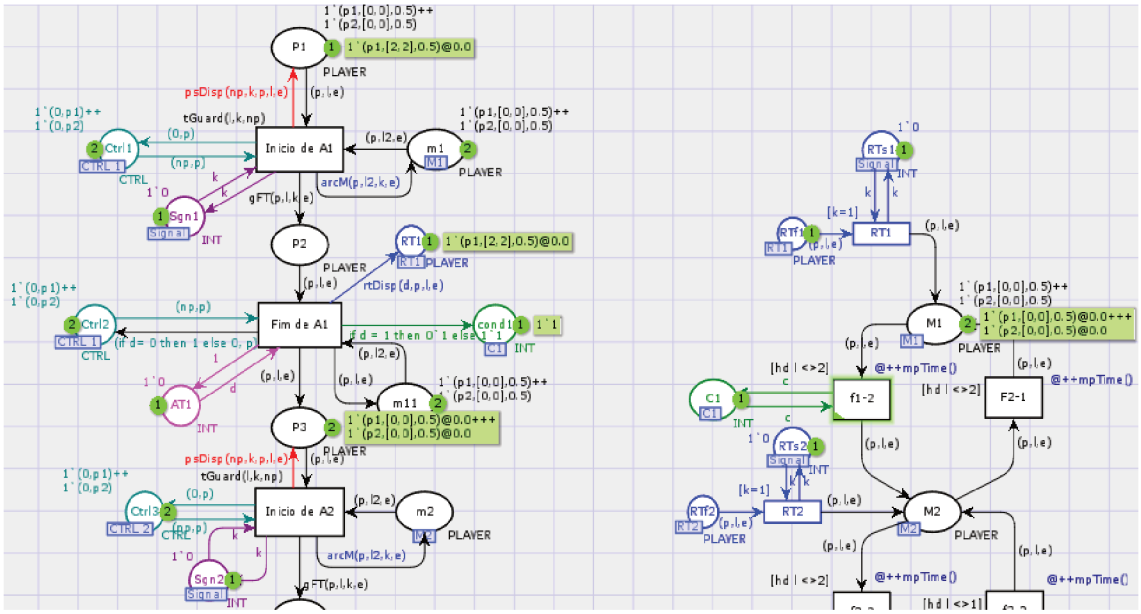


Figura 64 – Estado da rede após o disparo da transição *Fim de A1*.

Ao final da *quest* a variável *k* é atualizada indicando que a *quest* foi encerrada, e então as atividades associadas aos pseudo-disparos podem ser canceladas. Consequentemente, todas as transições de prefixo *RT* estarão habilitadas, como ilustra a figura 65. O disparo de *RT1* gera uma ficha no lugar *M1*, habilitando a transição *Inicio de A1* para o cancelamento do pseudo-disparo. O mesmo é feito para as demais transições do modelo que foram pseudo-disparadas.

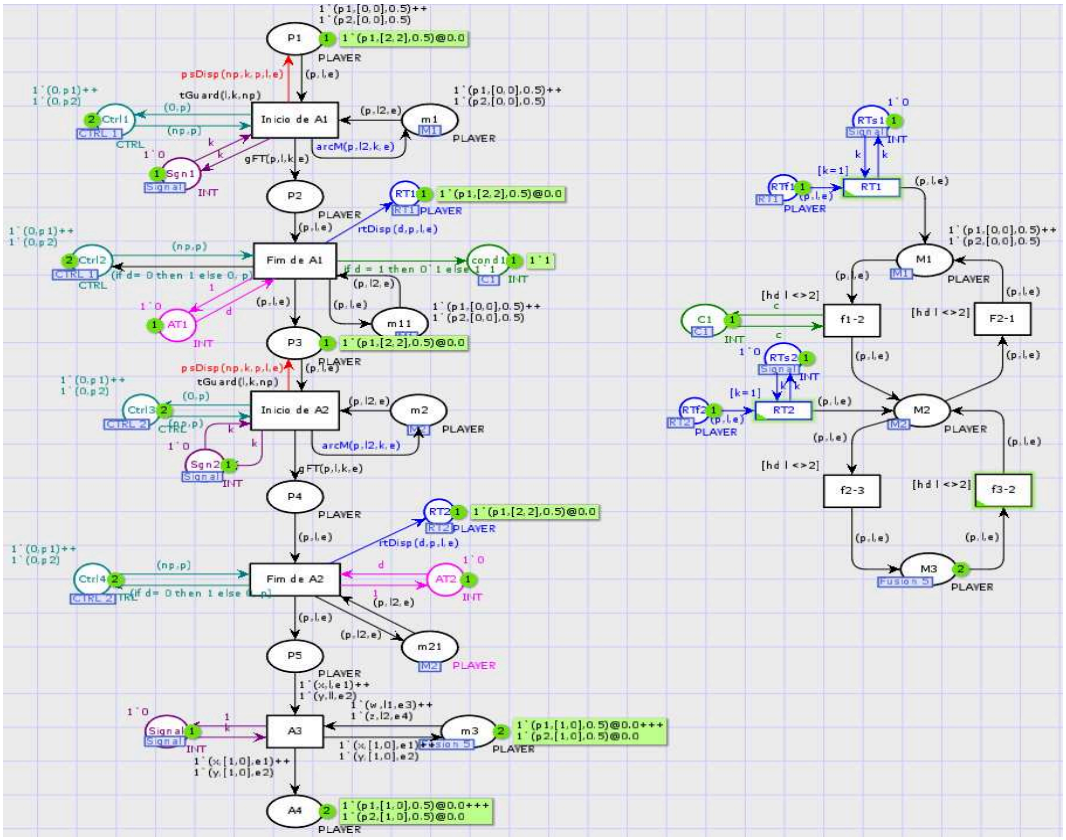


Figura 65 – Estado da rede ao final da *quest*.

O mecanismo de comunicação completo que estabelece as interações entre o Modelo Lógico e o Modelo Topológico no caso do modelo da *quest The Chemist* é apresentado na figura 66. O lado esquerdo da figura representa o Modelo Lógico Multiplayer, enquanto que o lado direito representa o Modelo Topológico Multiplayer. Todas as transições atividades são associadas a uma área do mapa topológico. As regiões de incerteza agregam os lugares especiais para gerenciar os disparos incertos com suas respectivas funções.

5.2.5 Análise do Modelo de Simulação Multiplayer

Um dos objetivos da criação de modelos de simulação, apresentados neste trabalho, é garantir que os estados de um jogo sejam sempre alcançáveis. No caso de um modelo lógico, um erro acontecerá quando uma atividade do jogo não puder, em nenhuma situação, ser executada pelo jogador. A verificação desse tipo de situação pode ser feita através da análise da propriedade de vivacidade, por exemplo. Se não há transições mortas no modelo, ou seja, a rede é livre de *deadlock*, é possível afirmar então que todas as atividades previstas no modelo do nível analisado poderão ser executadas eventualmente pelos jogadores.

Uma forma de verificar a propriedade de vivacidade é utilizando a funcionalidade de *state space* disponível no CPN Tools. O *state space* (espaço de estados) calcula todos os

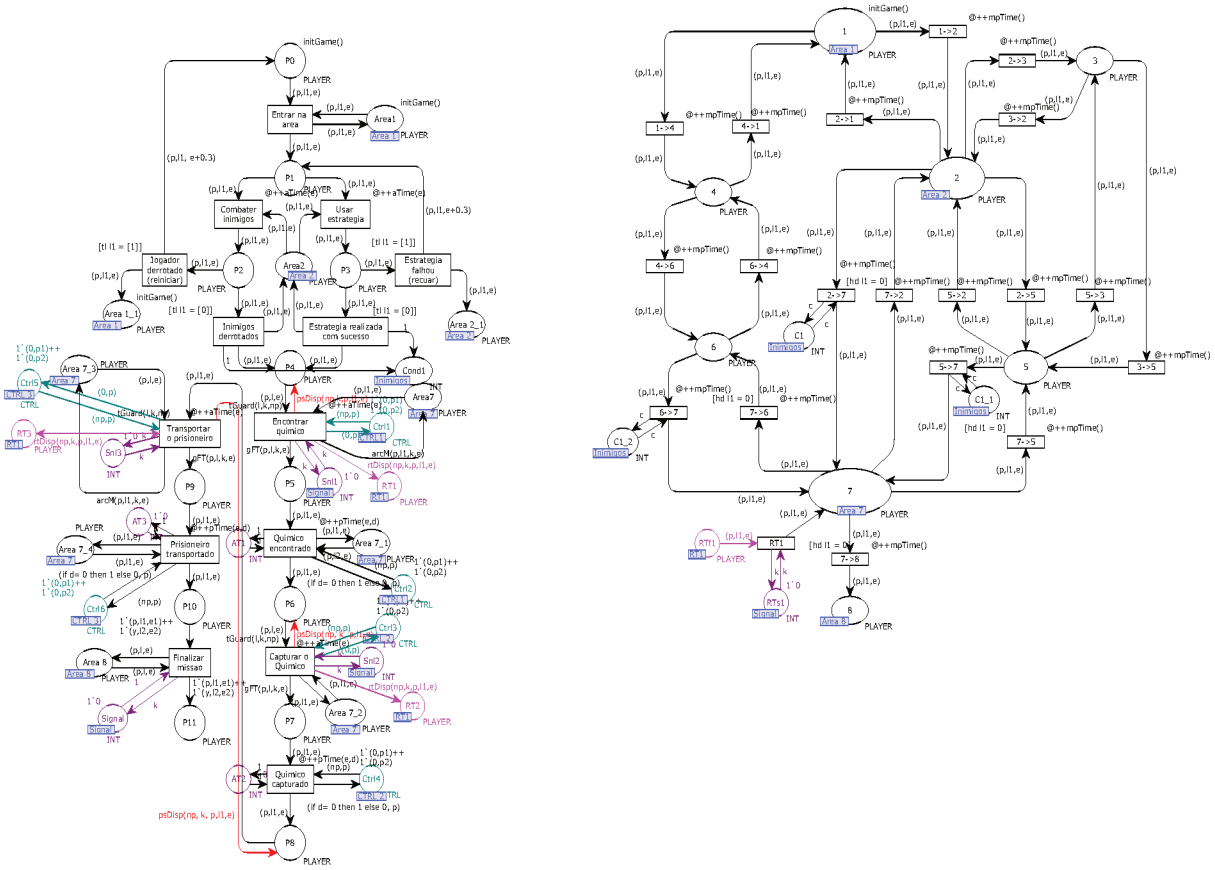


Figura 66 – Modelo Global da *quest The Chemist* do jogo Tom Clancy's Ghost Recon: Wildlands.

estados alcançáveis a partir da marcação inicial da rede de Petri. Com isso, a ferramenta gera um relatório com todas as propriedades obtidas a partir do cálculo do espaço de estado do modelo. Considere o exemplo da *quest The Chemist* citado na seção anterior e ilustrado na figura 67. Uma vez submetido ao *state space* no CPN Tools, o relatório de análise apresenta que não existem transições mortas no modelo a partir da marcação inicial. Sendo assim, é possível dizer que todas as atividades da *quest The Chemist* poderão ser executadas.

O mesmo tipo de análise pode ser realizada para o modelo topológico. Em um mundo virtual todas as áreas devem ser acessíveis para os jogadores; caso contrário um erro de concepção do jogo acontecerá. No caso do Modelo Topológico Multiplayer, o que indica o acesso das áreas são os disparos das transições. Então quando não houver situação de *deadlock* na rede significa que todas as áreas do mapa são acessíveis para os jogadores. Considere o mapa topológico da *quest The Chemist*. Ao ser submetido à análise do espaço de estados no CPN Tools o relatório de análise indica que não existem transições mortas no modelo. Em outras palavras, todas as áreas do mapa podem ser acessadas pelos jogadores. É importante ressaltar que para essa análise foi considerado apenas o mapa topológico sem os lugares que representam as condições para liberação de áreas que dependem do

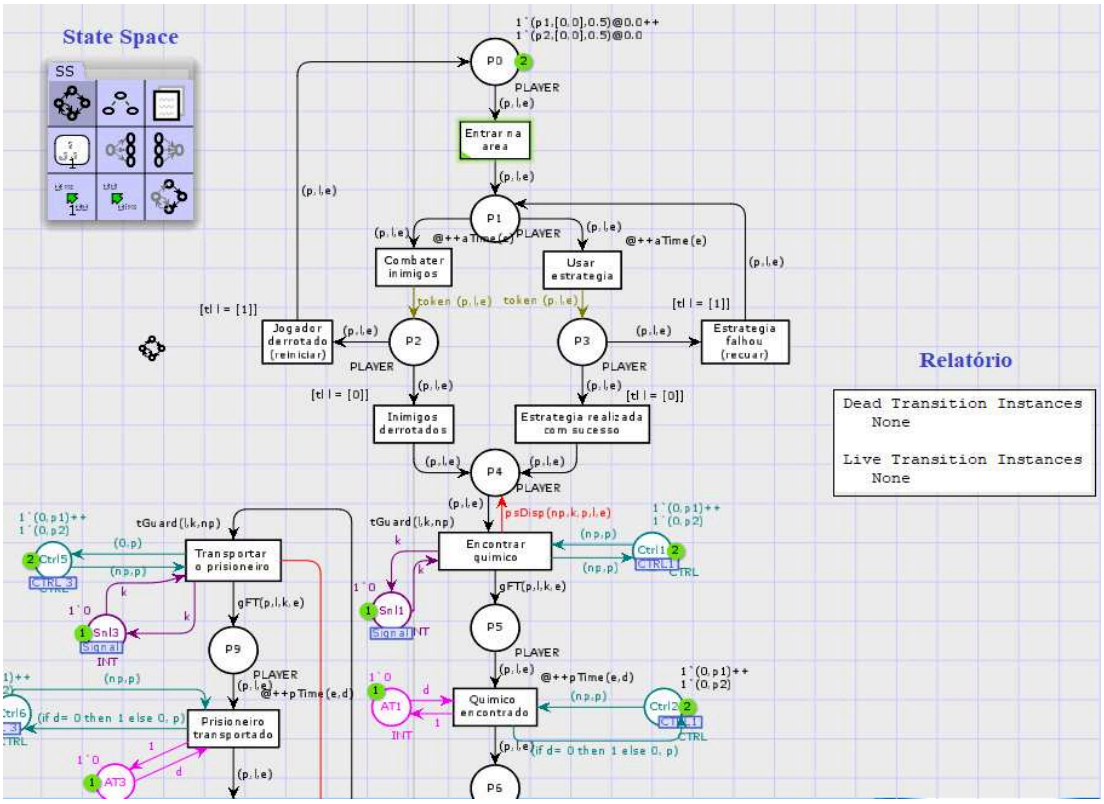


Figura 67 – *State space* para o Modelo Lógico Multiplayer da *quest The Chemist*.

modelo/mecanismo de comunicação entre o Modelo Lógico e o Modelo Topológico.

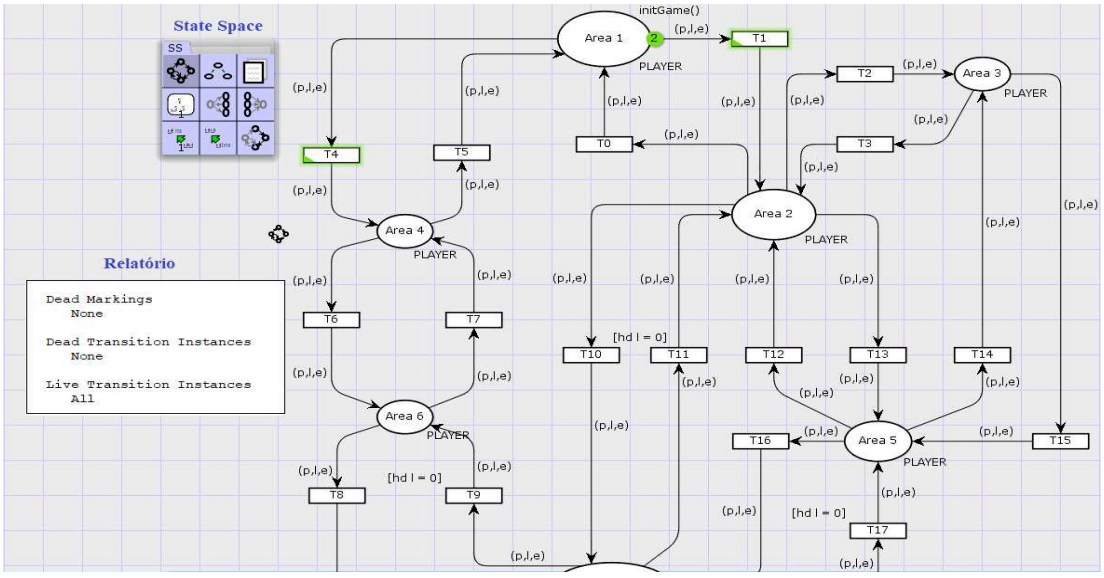


Figura 68 – State space para o Modelo Topológico Multiplayer da *quest The Chemist*.

Por meio do modelo de simulação também é possível realizar a estimação do tempo médio de jogo no caso multiplayer. A verificação do tempo médio de jogo no caso multiplayer, proposta neste trabalho, consiste em usar simulação automática por replicação. Cada modelo pode ser simulado separadamente e ter seu tempo mínimo, máximo e médio

de execução estimado. No CPN Tools a simulação por replicação é feita por meio da linha de comando *CPN'Replications.nreplications N*, onde *N* é o número de replicações que se deseja fazer.

Considere o exemplo do Modelo Lógico Multiplayer da *quest The Chemist*. Para este exemplo, a simulação foi feita com 10 replicações. O relatório de simulação é apresentado na figura 69. Para todas as 10 simulações o critério de parada foi por não haver mais transições habilitadas. Além disso, o número médio de passos (disparos de transições) para finalizar as simulações foi de 29 passos. O tempo mínimo para a execução da *quest* foi de aproximadamente 4.05 unidades de tempo, e o tempo máximo foi de 21 unidades de tempo. O tempo médio para a execução da *quest*, de acordo com as simulações, foi de 8 unidades de tempo.

```
Simulation no.: 1
Steps.........: 26
Model time....: 4.0544172428
Stop reason...: No more enabled transitions!
Time to run simulation: 0 seconds

Simulation no.: 2
Steps.........: 26
Model time....: 5.59197958473
Stop reason...: No more enabled transitions!
Time to run simulation: 0 seconds

Simulation no.: 3
Steps.........: 41
Model time....: 13.1025316032
Stop reason...: No more enabled transitions!
Time to run simulation: 0 seconds

Simulation no.: 4
Steps.........: 23
Model time....: 4.56012382848
Stop reason...: No more enabled transitions!
Time to run simulation: 0 seconds

Simulation no.: 5
Steps.........: 26
Model time....: 7.11326856102
Stop reason...: No more enabled transitions!
Time to run simulation: 0 seconds

Simulation no.: 6
Steps.........: 38
Model time....: 21.0754897506
Stop reason...: No more enabled transitions!
Time to run simulation: 0 seconds

Simulation no.: 7
Steps.........: 28
Model time....: 8.0253525448
Stop reason...: No more enabled transitions!
Time to run simulation: 0 seconds

Simulation no.: 8
Steps.........: 26
Model time....: 3.85693212384
Stop reason...: No more enabled transitions!
Time to run simulation: 0 seconds

Simulation no.: 9
Steps.........: 28
Model time....: 3.84977795721
Stop reason...: No more enabled transitions!
Time to run simulation: 0 seconds

Simulation no.: 10
Steps.........: 28
Model time....: 9.82346377341
Stop reason...: No more enabled transitions!
Time to run simulation: 0 seconds
```

Figura 69 – Relatório de simulação com 10 replicações do Modelo Lógico Multiplayer da *quest The Chemist*.

O mesmo procedimento de estimação do tempo médio pode ser aplicado ao Modelo Topológico Multiplayer. O modelo apresentado na figura 51 foi submetido a simulação com 10 replicações. O relatório de Simulação do Modelo Topológico Multiplayer da *quest The Chemist* pode ser visto na figura 70. Nesse caso, o número médio de passos para finalizar as simulações foi de aproximadamente 67 passos. O tempo mínimo para percorrer todas as áreas do mapa foi de aproximadamente 7.58 unidades de tempo, enquanto que o tempo máximo foi de aproximadamente 56 unidades de tempo. O tempo médio para os jogadores acessarem as áreas do mapa foi de aproximadamente 39 unidades de tempo.

Na abordagem proposta nesta pesquisa, um cenário de jogo multiplayer é definido pela representação do Modelo Global Multiplayer (Modelo Lógico Multiplayer + Modelo Topológico Multiplayer). Portanto, para estimar o tempo médio de um nível ou *quest* de um jogo multiplayer é preciso submeter o Modelo Global Multiplayer a uma simulação por

```
Simulation no.: 1
Steps.........: 25
Model time....: 7.58651708522
Stop reason...: No more enabled transitions!
Time to run simulation: 0 seconds

Simulation no.: 2
Steps.........: 20
Model time....: 10.829662984
Stop reason...: No more enabled transitions!
Time to run simulation: 0 seconds

Simulation no.: 3
Steps.........: 96
Model time....: 45.9519736819
Stop reason...: No more enabled transitions!
Time to run simulation: 0 seconds

Simulation no.: 4
Steps.........: 75
Model time....: 31.0175559055
Stop reason...: No more enabled transitions!
Time to run simulation: 0 seconds

Simulation no.: 5
Steps.........: 87
Model time....: 40.2466039536
Stop reason...: No more enabled transitions!
Time to run simulation: 0 seconds

Simulation no.: 6
Steps.........: 37
Model time....: 14.8215763442
Stop reason...: No more enabled transitions!
Time to run simulation: 0 seconds

Simulation no.: 7
Steps.........: 59
Model time....: 28.8146866876
Stop reason...: No more enabled transitions!
Time to run simulation: 0 seconds

Simulation no.: 8
Steps.........: 51
Model time....: 23.3132019337
Stop reason...: No more enabled transitions!
Time to run simulation: 0 seconds

Simulation no.: 9
Steps.........: 118
Model time....: 55.744084595
Stop reason...: No more enabled transitions!
Time to run simulation: 0 seconds

Simulation no.: 10
Steps.........: 97
Model time....: 49.3685973427
Stop reason...: No more enabled transitions!
Time to run simulation: 0 seconds
```

Figura 70 – Relatório de simulação com 10 replicações do Modelo Topológico Multiplayer da *quest The Chemist*.

meio de replicações. Como exemplo, considere o modelo global da *quest The Chemist*. De acordo com o relatório de simulação apresentado na figura 71, o tempo mínimo estimado para a execução da *quest* é de aproximadamente 10 unidades de tempo, enquanto que o tempo máximo é de aproximadamente 64,84 unidades de tempo. Já o tempo médio estimado para a execução da *quest* é de aproximadamente 31 unidades de tempo.

```
Simulation no.: 1
Steps.........: 83
Model time....: 31.9848794307
Stop reason...: No more enabled transitions!
Time to run simulation: 0 seconds

Simulation no.: 2
Steps.........: 136
Model time....: 64.8462961211
Stop reason...: No more enabled transitions!
Time to run simulation: 0 seconds

Simulation no.: 3
Steps.........: 34
Model time....: 10.1385358349
Stop reason...: No more enabled transitions!
Time to run simulation: 0 seconds

Simulation no.: 4
Steps.........: 151
Model time....: 46.0920802208
Stop reason...: No more enabled transitions!
Time to run simulation: 0 seconds

Simulation no.: 5
Steps.........: 64
Model time....: 20.527551119
Stop reason...: No more enabled transitions!
Time to run simulation: 0 seconds

Simulation no.: 6
Steps.........: 83
Model time....: 36.6146215569
Stop reason...: No more enabled transitions!
Time to run simulation: 0 seconds

Simulation no.: 7
Steps.........: 116
Model time....: 33.3011962659
Stop reason...: No more enabled transitions!
Time to run simulation: 0 seconds

Simulation no.: 8
Steps.........: 84
Model time....: 23.9868043416
Stop reason...: No more enabled transitions!
Time to run simulation: 0 seconds

Simulation no.: 9
Steps.........: 70
Model time....: 22.5552454492
Stop reason...: No more enabled transitions!
Time to run simulation: 0 seconds

Simulation no.: 10
Steps.........: 59
Model time....: 18.1636355637
Stop reason...: No more enabled transitions!
Time to run simulation: 0 seconds
```

Figura 71 – Relatório de simulação com 10 replicações do Modelo Global Multiplayer da *quest The Chemist*.

Em alguns jogos multiplayer, o número de jogadores pode ser determinante para a boa execução do jogo. Em outros casos, no entanto, a quantidade de jogadores que

compartilham um mesmo cenário não tem grande influência em sua execução. Por meio do modelo de simulação implementado no CPN Tools é possível analisar o papel de cada jogador na execução do cenário de jogo, do ponto de vista da jogabilidade. Para tanto, precisa-se mapear a execução de cada atividade do jogo a fim de determinar qual jogador concluiu qual atividade.

Uma forma de monitorar as atividades é por meio da implementação de Monitores. Um Monitor é uma funcionalidade do CPN Tools que permite observar, inspecionar, controlar ou modificar uma simulação. Os monitores podem inspecionar as marcações de lugares e as variáveis que ocorrem durante uma simulação e podem tomar as ações apropriadas com base nas observações obtidas. Na abordagem apresentada nesta pesquisa será utilizado um monitor do tipo coletor de dados para extrair e armazenar as informações necessárias a partir do disparo das transições.

No caso do modelo multiplayer, monitorar as transições atividades que compõe as regiões de incerteza é importante, pois são essas transições que representam o compartilhamento de atividades no jogo. Sendo assim, um monitor é adicionada a cada transição atividade do modelo que é compartilhada entre os jogadores. Um monitor é definido basicamente pelo seu tipo e predicado. Neste caso, o tipo é *Write in file*, que permite que a informação coletada seja armazenada em um arquivo texto. Já o predicado define a informação que será coletada, que neste caso é o identificador de cada jogador que executou a atividade monitorada. Uma vez que os monitores são adicionados em todas as transições compartilhadas, executa-se a simulação por meio de replicações. Com os dados obtidos é possível estabelecer quais atividades os jogadores fizeram e quantos jogadores foram necessários para executar o jogo/*quest*.

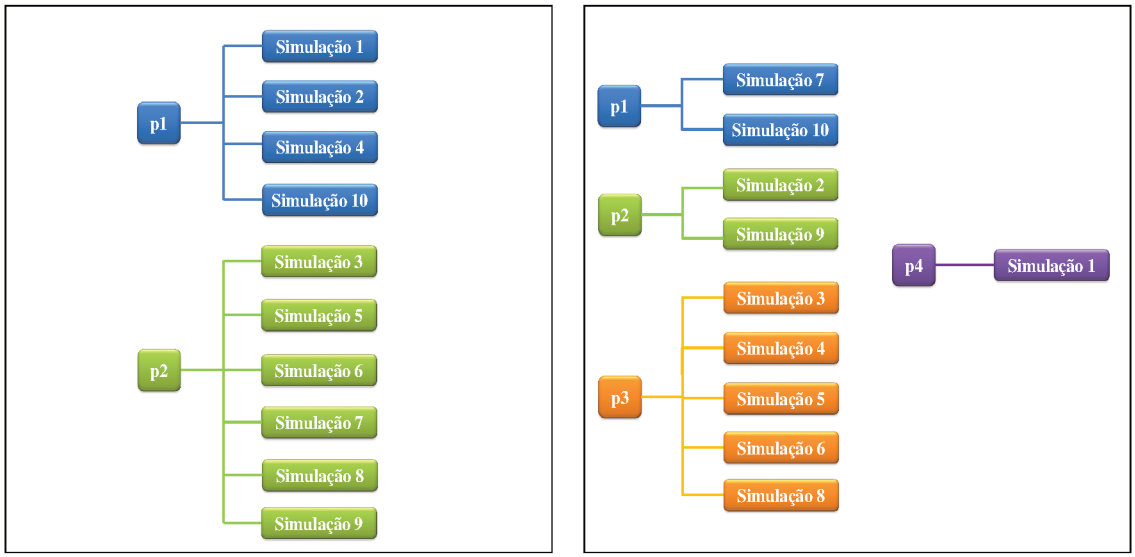
Para exemplificar essa abordagem, considere novamente o Modelo Global Multiplayer *quest The Chemist*. Neste exemplo as atividades que são compartilhadas e fazem parte da região de incerteza do modelo são: *Encontrar quimico* (E.Q.), *Quimico encontrado* (Q.E.), *Capturar quimico* (C.Q.), *Quimico capturado* (Q.C.), *Trasportar prisioneiro* (T.P.) e *Prisioneiro transportado* (P.T). Para cada uma dessas atividades, adicionou-se um monitor coletor de dados. Em seguida, uma simulação foi executada com 10 replicações. A fim de determinar se a quantidade de jogadores influencia na execução das atividades e no tempo de jogo, simulações foram realizadas para o modelo com 2, 4, 6, 8 e 10 jogadores.

A tabela 1 apresenta o resultado das simulações com 2 jogadores. De acordo com os resultados, quando 2 jogadores executam a *quest*, as atividades são realizadas de forma mais ou menos balanceada entre os jogadores. Por exemplo, o jogador *p1* desempenhou mais atividades em quatro de dez simulações, enquanto que o jogador *p2* desempenhou mais atividades em seis de dez simulações realizadas, como ilustra o gráfico da figura 72(a). Do ponto de vista de jogabilidade, os dois jogadores obtiveram desempenhos similares.

Os resultados das simulações com 4 jogadores são apresentados na tabela 2. No caso de 4 jogadores compartilhando a mesma *quest*, todos eles obtiveram desempenho em algum

Tabela 1 – Simulação por replicação para 2 jogadores.

	E.Q.	Q.E.	C.Q.	Q.C.	T.P.	P.T.
Simulação 1	p2	p2	p2	p2	p1	p1
Simulação 2	p2	p2	p1	p1	p2	p2
Simulação 3	p1	p1	p1	p2	p1	p1
Simulação 4	p2	p2	p1	p1	p1	p1
Simulação 5	p1	p1	p1	p1	p1	p1
Simulação 6	p1	p1	p1	p1	p1	p1
Simulação 7	p1	p1	p1	p1	p1	p1
Simulação 8	p1	p1	p1	p1	p2	p2
Simulação 9	p1	p1	p1	p1	p1	p1
Simulação 10	p2	p2	p2	p2	p2	p2



(a) Desempenho de 2 jogadores na *quest The Chemist* com 10 simulações.

(b) Desempenho de 4 jogadores na *quest The Chemist* com 10 simulações.

Figura 72 – Monitoramento da execução da *quest The Chemist* para cenários com 2 e 4 jogadores.

momento. No entanto, não na mesma proporção. Por exemplo, a figura 72(b) mostra que o jogador *p3* desempenhou mais atividades em cinco simulações, enquanto que o jogador *p4* desempenhou em apenas uma.

Já com 6 jogadores apenas três deles executaram atividades na *quest*, como ilustram a tabela 3 e figura 73(a).

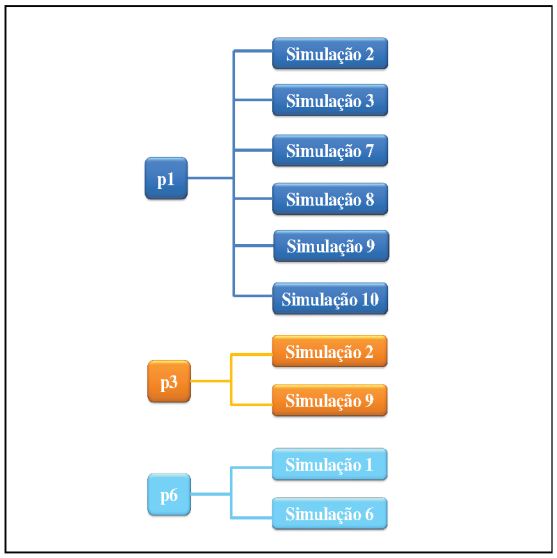
Por outro lado, no caso de 8 e 10 jogadores compartilhando a *quest*, apenas um deles não obteve desempenho, como ilustram as tabelas 4 e 5, e as figuras 73(b) e 74. De acordo com o resultado das replicações é possível perceber que a maioria dos jogadores participaram da *quest* executando pelo menos uma atividade. No entanto, acima de 4 jogadores se tornou mais comum a não participação de pelo menos um jogador na execução das atividades. Considerando que todas as atividades da *quest* analisada são por compartilhamento colaborativo, do ponto de vista de jogabilidade, esse tipo de comportamento

Tabela 2 – Simulação por replicação para 4 jogadores.

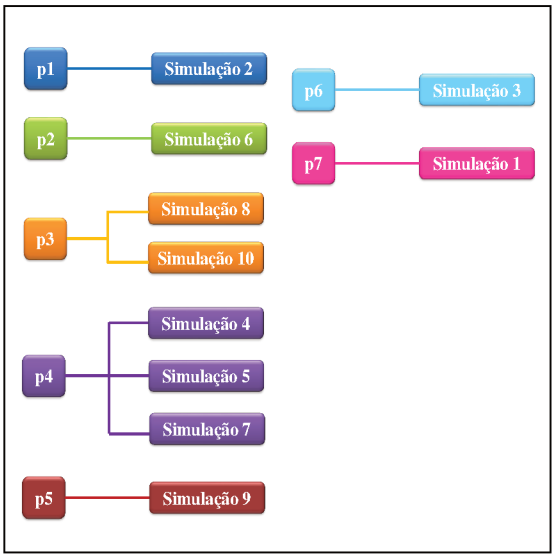
	E.Q.	**Q.E.**	**C.Q.**	**Q.C.**	**T.P.**	**P.T.**
Simulação 1	p4	p4	p3	p3	p3	p3
Simulação 2	p2	p2	p3	p3	p3	p3
Simulação 3	p3	p3	p3	p3	p2	p2
Simulação 4	p3	p3	p3	p3	p3	p3
Simulação 5	p3	p3	p3	p3	p3	p3
Simulação 6	p3	p3	p3	p3	p2	p2
Simulação 7	p1	p1	p3	p3	p4	p4
Simulação 8	p3	p3	p1	p1	p2	p2
Simulação 9	p2	p2	p2	p2	p4	p4
Simulação 10	p1	p1	p1	p1	p1	p1

Tabela 3 – Simulação por replicação para 6 jogadores.

	E.Q.	**Q.E.**	**C.Q.**	**Q.C.**	**T.P.**	**P.T.**
Simulação 1	p6	p6	p5	p5	p3	p3
Simulação 2	p1	p1	p5	p5	p5	p5
Simulação 3	p1	p1	p1	p1	p1	p1
Simulação 4	p3	p3	p5	p5	p4	p4
Simulação 5	p3	p3	p5	p5	p2	p2
Simulação 6	p6	p6	p5	p5	p1	p1
Simulação 7	p1	p1	p6	p6	p6	p6
Simulação 8	p1	p1	p4	p4	p6	p6
Simulação 9	p1	p1	p3	p3	p2	p2
Simulação 10	p1	p1	p5	p5	p5	p5



(a) Desempenho de 6 jogadores na *quest The Chemist* com 10 simulações.



(b) Desempenho de 8 jogadores na *quest The Chemist* com 10 simulações.

Figura 73 – Monitoramento da execução da *quest The Chemist* para cenários com 6 e 8 jogadores.

indica que há jogadores que não estão contribuindo na realização do objetivo do jogo. Porém, a estrutura do jogo é adequada para o caso multiplayer.

Tabela 4 – Simulação por replicação para 8 jogadores.

	E.Q.	**Q.E.**	**C.Q.**	**Q.C.**	**T.P.**	**P.T.**
Simulação 1	p7	p7	p7	p7	p6	p6
Simulação 2	p1	p1	p4	p4	p7	p7
Simulação 3	p6	p6	p2	p2	p2	p2
Simulação 4	p4	p4	p8	p8	p2	p2
Simulação 5	p4	p4	p8	p8	p4	p4
Simulação 6	p2	p2	p4	p4	p7	p7
Simulação 7	p4	p4	p7	p7	p1	p1
Simulação 8	p3	p3	p8	p8	p1	p1
Simulação 9	p5	p5	p8	p8	p1	p1
Simulação 10	p3	p3	p5	p5	p8	p8

Tabela 5 – Simulação por replicação para 10 jogadores.

	E.Q.	**Q.E.**	**C.Q.**	**Q.C.**	**T.P.**	**P.T.**
Simulação 1	p9	p9	p9	p9	p10	p10
Simulação 2	p4	p4	p4	p4	p8	p8
Simulação 3	p8	p8	p6	p6	p3	p3
Simulação 4	p7	p7	p4	p4	p6	p6
Simulação 5	p5	p5	p9	p9	p6	p6
Simulação 6	p2	p2	p2	p2	p1	p1
Simulação 7	p1	p1	p6	p6	p5	p5
Simulação 8	p6	p6	p8	p8	p3	p3
Simulação 9	p10	p10	p2	p2	p3	p3
Simulação 10	p6	p6	p5	p5	p3	p3

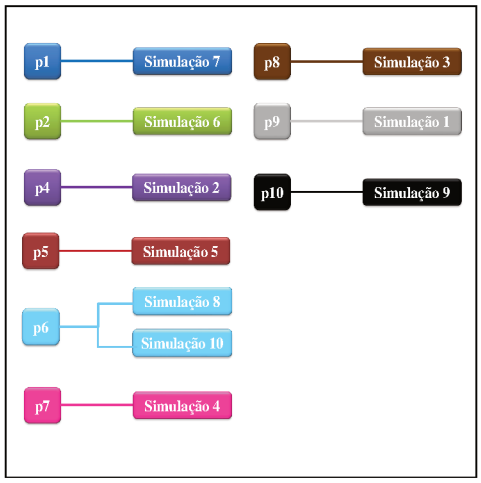


Figura 74 – Desempenho de 10 jogadores na *quest The Chemist* com 10 simulações.

Além de verificar a execução de cada atividade, também é possível avaliar o tempo de jogo para diferentes cenários. Neste caso, a simulação por replicação pode ser aplicada

para os modelos temporizados. A avaliação da performance do modelo multiplayer pode considerar cenários com no mínimo 2 e no máximo N jogadores. Para exemplificar esta abordagem, o modelo lógico e o modelo global multiplayer definidos para a *quest The Chemist* foram utilizados, considerando cenários com 2, 4, 6, 8 e 10 jogadores ativos ao mesmo.

A tabela 6 representa os tempos para a execução das atividades do Modelo Lógico Multiplayer. A primeira coluna ilustra os tempos mínimo, máximo e médio para um cenário com 2 jogadores, a segunda coluna apresenta os tempos para 4 jogadores e assim sucessivamente até a última coluna que apresenta os tempos para um cenário com 10 jogadores. No cenário com 2 jogadores o tempo médio da *quest* é de 8,78 unidades de tempo. Já no cenário com 6 jogadores o tempo médio aumenta para 9,83, assim como no último cenário onde o tempo médio é de 14,03 unidades tempo.

Tabela 6 – Resultados de simulação com 10 replicações de diferentes cenários para o Modelo Lógico Multiplayer.

	2 jogadores	**4 jogadores**	**6 jogadores**	**8 jogadores**	**10 jogadores**
Mínimo	4,05	3,32	3,36	8,16	9,11
Máximo	18,30	18,23	16,29	21,08	13,59
Médio	8,78	7,60	9,83	12,37	14,03

O gráfico ilustrado na figura 75 apresenta os tempos de execução da *quest* de acordo com os cenários simulados. É possível perceber que o aumento do número de jogadores leva ao aumento no tempo de execução das atividades. No caso da *quest* analisada nesse exemplo esse comportamento é esperado, pois as atividades são colaborativas, fazendo com que os jogadores dependam uns dos outros. Em outras palavras, mesmo que um jogador execute sozinho todas as atividades em um período curto de tempo, ele precisará, em algum momento do jogo, esperar pela chegada dos demais jogadores no mesmo estado em que ele se encontra para prosseguir com as outras atividades do jogo. Dessa forma, é natural que ao aumentar o número de jogadores, consequentemente, aumenta-se o tempo de execução da *quest*/jogo.

É importante também simular o cenário de jogo com a inserção do mapa topológico. Para tanto, o Modelo Global Multiplayer é utilizado. A tabela 7 apresenta os resultados de simulação com 10 replicações para o Modelo Global Multiplayer da *quest The Chemist*, considerando diferentes cenários. Neste caso, o tempo de execução é maior devido a interação do modelo lógico com o modelo topológico. Além disso, de acordo com o aumento do número de jogadores, o tempo de execução do cenário global também aumenta. Por exemplo, com 2 jogadores o tempo médio é de 30,82 unidades de tempo, já para 10 jogadores o tempo médio de execução é de 47,03 unidades de tempo. A comparação entre os tempos de execução do Modelo Global Multiplayer de acordo com o número de jogadores pode ser visto no gráfico ilustrado na figura 76.

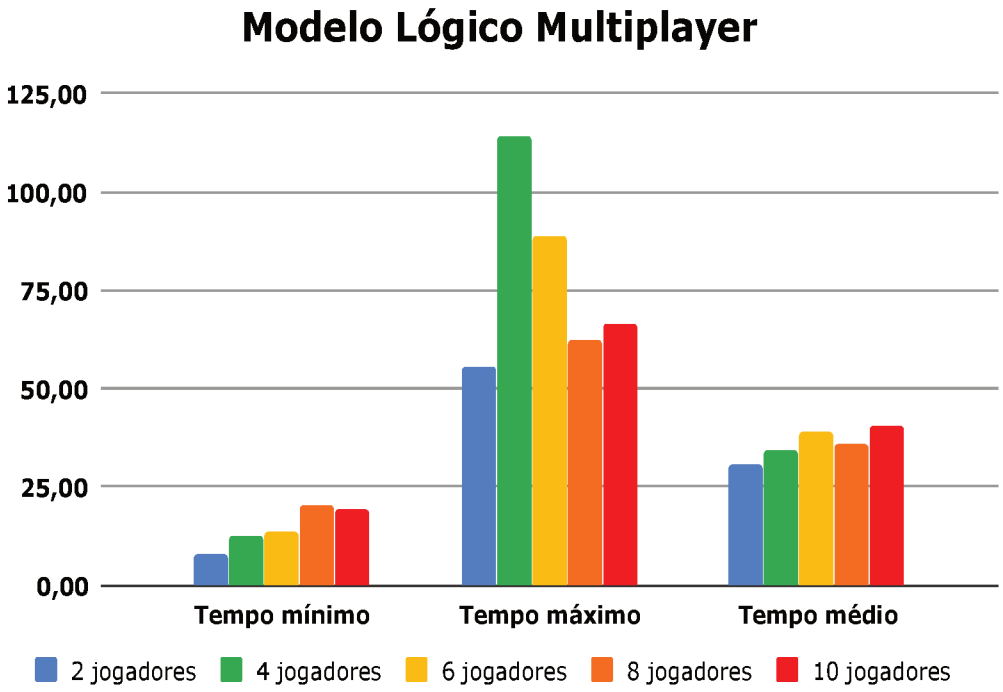


Figura 75 – Comparação do tempo de execução mínimo, máximo e médio para o Modelo Lógico Multiplayer com 2, 4, 6, 8 e 10 jogadores.

Tabela 7 – Resultados de simulação com 10 replicações de diferentes cenários para o Modelo Global Multiplayer.

	2 jogadores	**4 jogadores**	**6 jogadores**	**8 jogadores**	**10 jogadores**
Mínimo	10,13	12,48	30,54	21,46	36,21
Máximo	64,84	69,28	62,20	63,62	90,50
Médio	30,82	34,69	44,19	39,84	47,03

5.3 Considerações Finais

Nesse capítulo foi apresentado uma abordagem para a representação de cenários de jogos multiplayer baseada nas WorkFlow nets Possibilísticas e nos grafos de estados. O conceito de WorkFlow net Possibilística foi utilizado para definir o Modelo Lógico Multiplayer que representa as atividades existentes em um jogo. O uso de uma WorkFlow net Possibilística permitiu a representação do comportamento esperado de um jogador em um jogo, bem como o comportamento inesperado gerado pelas escolhas individuais de jogadores distintos. Já a rede de Petri do tipo grafo de estados foi utilizada para definir o Modelo Topológico Multiplayer que representa as áreas significativas do mapa, formando então o mapa topológico do mundo virtual. Para representar um cenário de jogo multiplayer, foi definido então um mecanismo de comunicação para estabelecer a relação entre o modelo lógico e topológico multiplayer.

Também nesse capítulo foi proposta uma abordagem para a implementação do modelo

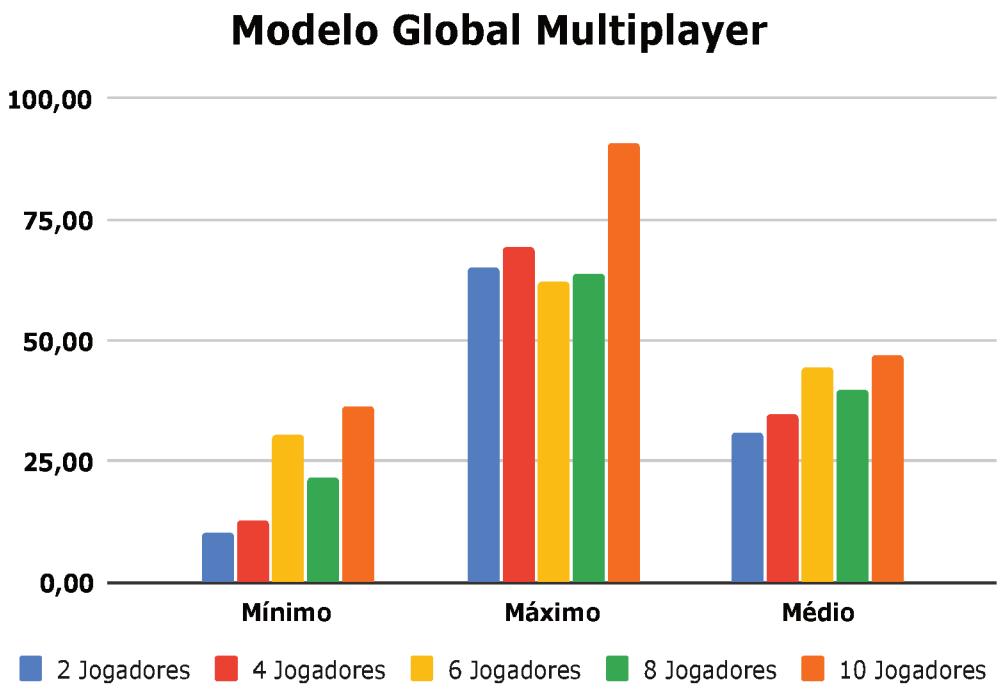


Figura 76 – Comparação do tempo de execução mínimo, máximo e médio para o Modelo Global Multiplayer com 2, 4, 6, 8 e 10 jogadores.

de simulação para cenários de jogos multiplayer. O modelo de simulação foi implementado no software CPN Tools. A implementação do modelo usando o CPN Tools permitiu programar funções para representar o funcionamento dos modelos quando inserida a noção de incerteza. Por meio de funcionalidades de simulação da ferramenta, também foi possível simular o modelo repetidas vezes a fim de verificar erros no fluxo de atividades e no acesso as áreas do mapa topológico, que podem comprometer o bom funcionamento do jogo, em termos de jogabilidade.

A replicação da simulação do modelo também é interessante para o estudo da jogabilidade do jogo ainda nos estágios iniciais de criação. Para isso, essa pesquisa propôs o uso de monitores como ferramenta para mapear a execução das atividades. Com isso, foi possível identificar o papel de cada jogador na realização das atividades do jogo. Por exemplo, o estudo de jogabilidade pode mostrar que a estrutura de um jogo é mais adaptada ao caso singleplayer quando a maioria das atividades ficam realizadas por um mesmo jogador. Por outro lado, se a carga de cada jogador durante a *quest* é mais ou menos equivalente, em termos de quantidade de atividades executadas, então a realização do jogo será mais adaptada ao caso multiplayer. Além disso, por meio do modelo de simulação também é possível estimar o tempo de *quests* de vídeo games do tipo multiplayer.

Capítulo 6

Conclusão

Este capítulo apresenta as conclusões a respeito deste trabalho de pesquisa. A seção 6.1 mostra as principais contribuições. Na seção 6.2 são apresentados os trabalhos futuros que serão desenvolvidos com a continuidade desta pesquisa. E, por fim, a seção 6.3 lista as contribuições bibliográficas obtidas por meio deste trabalho de pesquisa.

6.1 Principais Contribuições

O presente trabalho apresenta métodos para a modelagem, análise e verificação formal de cenários de vídeo games *singleplayer* e *multiplayer* utilizando para isso o formalismo das redes de Petri. Em particular, são utilizados diferentes tipos de redes de Petri para expressar de maneira eficiente os diferentes aspectos encontrados nos cenários de vídeo games.

O capítulo 4 apresenta um método, baseado em WorkFlow nets e grafos de estados, capaz de descrever as características principais de cenários de vídeo games do tipo singleplayer. Para tanto, o método considera que um jogo pode ser dividido em diferentes níveis e que cada nível possui um conjunto de atividades que precisam ser executadas no mundo virtual do jogo. Dessa forma, um modelo capaz de representar a sequência lógica das atividades de jogo pode ser definido. Também pode ser definido um modelo para a representação das áreas do mapa do mundo virtual onde são executadas as atividades do nível correspondente. Com a definição do Modelo Lógico e Modelo Topológico para cenários de vídeos do tipo singleplayer, tem-se que a primeira hipótese desta pesquisa foi validada, isto é, *usando modelos formais baseados na teoria das redes de Petri, especificamente os tipos especiais de redes de Petri como as WorkFlow nets e os grafos de estados, é possível descrever as características principais do fluxo de atividades existente em um vídeo game, bem como a topologia do mundo virtual.*

O método de análise proposto no capítulo 4 permite identificar, por meio das boas propriedades das redes de Petri, a corretude do jogo em termos de jogabilidade. Trata-se de um método de análise qualitativa em termo de jogabilidade de um modelo autônomo

não temporizado (correta execução dos cenários de jogo de cada nível considerando o mapa topológico do jogo). Portanto, conclui-se que a segunda hipótese desta pesquisa, também foi validada, ou seja, *é possível especificar um método de verificação e validação formal baseado nas boas propriedades das redes de Petri que pode ser aplicado à cenários de vídeo games, de tal modo a garantir o bom funcionamento do jogo em termos de sua jogabilidade.*

O capítulo 4, na seção 4.2, apresenta uma versão temporizada do modelo de jogo. A definição dos tempos de atividade ou de deslocamento para cada modelo foi possível por meio do uso de funções de tempo aleatórios associadas às transições dos modelos. Por meio de simulações automáticas (replicações) do modelo temporizado de jogo no CPN Tools torna-se possível estimar os tempos de jogo de cada nível. Por se tratar de dois modelos distintos que interagem (modelo lógico e topológico), a simulação estabelece em particular a influência que um modelo tem sobre o outro no cálculo dos tempos de jogo de cada nível. Considerando o método apresentado no capítulo 4, a terceira hipótese desta pesquisa foi validade, isto é, *as atividades de um vídeo game, incluindo a capacidade do jogador de percorrer vários lugares do mundo virtual, possuem, individualmente, um valor de duração associado. Tal valor pode ser representando em uma rede de Petri utilizando o conceito de temporização, em particular associando funções de tempo aleatório aos elementos ativos do modelo. Tais funções podem ter como parâmetro o nível de experiência/habilidade do jogador e assim gerar durações de atividades consistentes com a realidade de cada jogador.*

O método apresentado no capítulo 5 tem como objetivo validar a quarta hipótese desta pesquisa: *utilizando modelos baseado em redes de Petri é possível representar de maneira efetiva o comportamento dos cenários de vídeo games multiplayer por meio do conceito de incerteza inserido em uma versão Possibilística das redes de Petri, explorando então os conceitos de marcações imprecisas e disparos incertos para o modelo de atividades e de representação topológica.* O método propõe um modelo lógico possibilístico com possibilidade de cancelamento de atividades que permite a representação de cenário de jogos no caso *multiplayer* (2 jogadores ou mais que colaboram). O modelo proposto é baseado nas WorkFlow net Possibilísticas. O conceito de WorkFlow net Possibilística permite a representação de atividades compartilhadas entre os jogadores. De fato, num contexto de jogo *multiplayer* colaborativo, somente a noção de possibilidade de execução de cenário pode ser associada a cada jogador, já que não fica claro quais atividades cada jogador vai executar quando se inicia um nível de jogo. A mesma noção de incerteza não é associada ao mapa topológico, uma vez que a localização física do jogador durante o jogo não é passível de imprecisão.

Finalmente, o método para análise da jogabilidade de cenário de vídeo games multiplayer, em particular em relação à corretude da arquitetura do jogo e a estimação do tempo de jogo global quando se considera um grupo de jogador em vez de um jogador único, foi apresentado na seção 5.2.5. Este método considera, inicialmente, a verificação

dos modelos lógico e topológico multiplayer separadamente usando a análise do espaço de estados. Com essa análise é possível verificar propriedades das redes de Petri que podem garantir a corretude da estrutura da *quest* e do mapa topológico correspondente. Em seguida, considera-se o mesmo tipo de análise para o Modelo Global Multiplayer, a fim de verificar com a interação dos dois modelos distintos, por meio do mecanismo de comunicação, a corretude do cenário de jogo multiplayer. Tal cenário verifica se no cenário de jogo os jogadores podem executar todas as atividades e ter acesso a todas as áreas do mapa. O método para o estudo da jogabilidade utiliza o monitoramento das atividade e a simulação por replicação para diferentes cenários. Dessa forma, é possível estimar o tempo de jogo multiplayer bem como o papel de cada jogador no jogo. Assim, tem-se a validação da quinta hipótese desta pesquisa, isto é, *considerando os modelos construídos sob a perspectiva das rede de Petri Possibilísticas, bem como a inserção de temporização em tais modelos, é factível a apresentação de um método de verificação e validação formal que pode ser aplicado a cenários de vídeo games do tipo multiplayer, com o objetivo de garantir o bom funcionamento do jogo em termos de sua jogabilidade. Para tanto, é possível considerar as propriedades estruturais das redes de Petri e, além disso, se basear em ferramentas que permitem a implementação, análise e simulação de modelos baseados na teoria das redes de Petri.*

6.2 Trabalhos Futuros

Como trabalho futuro, deve-se investigar como as regras de jogo podem ser utilizadas com o propósito de aprimorar o modelo de simulação para representar aspectos importantes que determinam situações reais na execução do jogo. Por exemplo, incluir características dos jogadores que podem ser alteradas conforme à evolução do jogo como pontos de vida, itens, níveis de habilidades, entre outros.

Além disso, pode-se considerar uma abordagem hierárquica para a representação das *quests* do jogo, formando assim um conjunto de *quests* que podem ser analisadas separadamente, ou em conjunto, a partir da perspectiva global do jogo. Esse tipo de análise pode ser particularmente interessante para identificar problemas da narrativa do jogo, antes de ser implementado.

6.3 Contribuições em Produção Bibliográfica

As produções bibliográficas resultantes desta pesquisa são apresentadas a seguir:

- Artigo intitulado *Modeling of Video Games Using Workflow Nets and State Graphs*, publicado no *31st Brazilian Symposium on Software Engineering* em 2017 (BAR-

RETO; JULIA, 2017). Este artigo refere-se ao método apresentado no capítulo 4.

- Artigo intitulado *A Timed Petri Net Model to Specify Scenarios of Video Games*, publicado no *15th International Conference on Information Technology : New Generations* em 2018 (BARRETO; FREITAS; JULIA, 2018). Este artigo refere-se ao método apresentado na seção 4.2.

- Artigo intitulado *An Approach Based on Possibilistic WorkFlow Nets to Model Multiplayer Video Games*, publicado no *15th International Conference on Information Technology : New Generations* em 2018 (BARRETO; REZENDE; JULIA, 2018). Este artigo refere-se ao método apresentado no capítulo 5.

- Artigo intitulado *A Possibilistic Simulation Model for Multiplayer Game Scenarios Using CPN Tools*, aceito para ser publicado no *34th Brazilian Symposium on Software Engineering* em 2020 (BARRETO; JULIA, 2020). Este artigo refere-se ao método apresentado na seção 5.2.

Referências

Marcos Silvano Orita Almeida and Flavio Soares Correa da Silva. 2013. A systematic review of game design methods and tools. In *International Conference on Entertainment Computing*. Springer Berlin Heidelberg, Berlin, Heidelberg. <https://doi.org/10.1007/978-3-642-41106-9_3>

D Andreu, JC Pascal, and R Valette. 1996. Events as a key of a batch process control system. In *Symposium on discrete events and manufacturing systems (Lille, July 9-12, 1996)*. Gerf EC Lille, Villeneuve d'Ascq, Lille, FR.

Chee Siang Ang and G. S. V. Radha Krishna Rao. 2004. Designing Interactivity in Computer Games: a UML Approach. *Int. J. Intell. Games Simulation* (2004).

Manuel Araújo and Licínio Roque. 2009. Modeling Games with Petri Nets. In *DiGRA - Proceedings of the 2009 DiGRA International Conference: Breaking New Ground: Innovation in Games, Play, Practice and Theory*. Brunel University, Kingston, London.

Franciny M Barreto, Joslaine Cristina Jeske de Freitas, and Stéphane Julia. 2018a. A Timed Petri Net Model to Specify Scenarios of Video Games. In *Information Technology-New Generations*. Springer, Cham. <https://doi.org/10.1007/978-3-319-77028-4_61>

Franciny M Barreto, Leiliane Pereira de Rezende, and Stéphane Julia. 2018b. An Approach Based on Possibilistic WorkFlow Nets to Model Multiplayer Video Games. In *Information Technology-New Generations*. Springer, Cham. <https://doi.org/10.1007/978-3-319-77028-4_66>

Franciny M. Barreto and Stéphane Julia. 2014. Modeling and Analysis of Video Games Based on Workflow Nets and State Graphs. In *Proceedings of 24th Annual International Conference on Computer Science and Software Engineering*. IBM Corp., Riverton, NJ, USA. <https://doi.org/10.1145/3131151.3131193>

Franciny M Barreto and Stéphane Julia. 2017. Modeling of Video Games Using Workflow Nets and State Graphs. In *Proceedings of the 31st Brazilian Symposium on Software Engineering*. ACM, New York, NY, USA. <https://doi.org/10.1145/3131151.3131193>

Franciny M Barreto and Stéphane Julia. 2020. A Possibilistic Simulation Model for Multiplayer Game Scenarios Using CPN Tools. In *Proceedings of the 34st Brazilian Symposium on Software Engineering*. ACM, New York, NY, USA.

David Callele, Eric Neufeld, and Kevin Schneider. 2005. Requirements engineering and the creative process in the video game industry. In *Requirements Engineering, 2005. Proceedings. 13th IEEE International Conference on*. IEEE, Paris. <https://doi.org/10.1109/RE.2005.58>

Janette Cardoso and Robert Valette. 1997. *Redes de Petri*. Editora da UFSC, Florianópolis, SC.

Janette Cardoso, Robert Valette, and Didier Dubois. 1991. Petri nets with uncertain markings. In *Advances in Petri Nets 1990*. Springer Berlin Heidelberg, Berlin, Heidelberg.

Frédéric Collé, Ronan Champagnat, and Armelle Prigent. 2005. Scenario Analysis Based on Linear Logic. In *Proceedings of the 2005 ACM SIGCHI International Conference on Advances in Computer Entertainment Technology*. ACM, New York, NY, USA. <https://doi.org/10.1145/1178477.1178583>

Chris Crawford. 1984. *The art of computer game design*. Osborne/McGraw-Hill Berkeley, CA, Pullman, Washington.

René David and Hassane Alla. 1994. Petri nets for modeling of dynamic systems: A survey. *Automatica* (1994). <https://doi.org/10.1016/0005-1098(94)90024-8>

René David and Hassane Alla. 2010. *Discrete, continuous, and hybrid Petri nets*. Springer, Berlin, Heidelberg. <https://doi.org/10.1007/978-3-642-10669-9>

Guilherme Willian de Oliveira, Stéphane Julia, and Ligia Maria Soares Passos. 2011. Game modeling using WorkFlow nets. In *2011 IEEE International Conference on Systems, Man, and Cybernetics*. IEEE, Anchorage, AK. <https://doi.org/10.1109/ICSMC.2011.6083757>

Leiliane Pereira de Rezende, Stéphane Julia, and Janette Cardoso. 2012. Possibilistic workflow nets to deal with non-conformance in process execution. In *2012 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*. IEEE, Seoul, KOR. <https://doi.org/10.1109/ICSMC.2012.6377898>

Leiliane Pereira de Rezende., Stéphane Julia., and Janette Cardoso. 2016. Possibilistic WorkFlow Nets for Dealing with Cancellation Regions in Business Processes. In *Proceedings of the 18th International Conference on Enterprise Information Systems - Volume 2: ICEIS*. SciTePress, Rome, Italy. <https://doi.org/10.5220/0005760301260133>

Bernd Finkbeiner and Ernst-Rüdiger Olderog. 2017. Petri games: Synthesis of distributed systems with causal memory. *Information and Computation* (2017). <https://doi.org/10.1016/j.ic.2016.07.006>

Viviane Gal, Cécile Le Prado, Stéphane Natkin, and Liliana Vega. 2002. Writing for video games. *Proceedings Laval Virtual (IVRC)* (2002).

Johan Huizinga. 2000. *Homo Ludens*. Editora Perspectiva S.A., São Paulo, SP.

Kurt Jensen and Lars M Kristensen. 2009. *Coloured Petri nets: modelling and validation of concurrent systems*. Springer Science & Business Media, Berlin, Heidelberg. <https://doi.org/10.1007/b95112>

Hisham M. Kanode, Christopher M. e Haddad. 2009. Software Engineering Challenges in Game Development. In *ITNG*. IEEE Computer Society, Las Vegas, NV. <https://doi.org/10.1109/ITNG.2009.74>

Lars M Kristensen, Soren Christensen, and Kurt Jensen. 1998. The practitioner's guide to coloured Petri nets. *International Journal on Software Tools for Technology Transfer (STTT)* (1998). <https://doi.org/10.1007/s100090050021>

Young-Seol Lee and Sung-Bae Cho. 2012. Dynamic quest plot generation using Petri net planning. In *Proceedings of the Workshop at SIGGRAPH Asia*. Association for Computing Machinery, New York, NY, USA. <https://doi.org/10.1145/2425296.2425304>

Chris Lewis and Jim Whitehead. 2011. The whats and the whys of games and software engineering. In *Proceedings of the 1st international workshop on games and software engineering*. Association for Computing Machinery, New York, NY, USA. <https://doi.org/10.1145/1984674.1984676>

Robin Milner, Mads Tofte, Robert Harper, and David MacQueen. 1997. *The definition of standard ML: revised*. MIT press, Cambridge, MA, USA. <https://doi.org/10.7551/mitpress/2319.001.0001>

Tadao Murata. 1989. Petri nets: Properties, analysis and applications. *Proc. IEEE* (1989). <https://doi.org/10.1109/5.24143>

Stéphane Natkin, Liliana Vega, and S Grünvogel. 2004. A new methodology for spatiotemporal game design. In *Mehdi, Q. and Gough, N.,(Eds.). Proceedings of CGAIDE 2004, 5th Game-On International Conference on Computer Games: Artificial Intelligence, Design and Education*. The University of Wolverhampton, School of Computing and Information Technology, Wolverhampton, UK.

Katharine Neil. 2012. Game design tools: Time to evaluate. *Proceedings of 2012 DiGRA Nordic* (2012). <http://www.digra.org/wp-content/uploads/digital-library/12168.46494.pdf>

Jorge Peña, Bin Wu, Jordi Arranz, and Arne Traulsen. 2016. Evolutionary games of multiplayer cooperation on graphs. *PLoS computational biology* (2016). <https://doi.org/10.1371/journal.pcbi.1005059>

James Lyle Peterson. 1981. *Petri net theory and the modeling of systems*. Prentice-Hall, Englewood Cliffs, NJ, USA.

Carl Adam Petri. 1962. *Kommunikation mit Automaten*. Ph.D. Dissertation. Universität Hamburg, Germany.

Christian Reuter, Stefan Göbel, and Ralf Steinmetz. 2015. Detecting structural errors in scene-based Multiplayer Games using automatically generated Petri Nets. *Foundations of Digital Games, Pacific Grove, USA* (2015).

Emanuel Montero Reyno and José Á. Carsí Cubel. 2009. Automatic prototyping in model-driven game development. *Computers in Entertainment* (2009). <https://doi.org/10.1145/1541895.1541909>

Ruslan Rezin, Ilya Afanasyev, Manuel Mazzara, and Victor Rivera. 2018. Model Checking in multiplayer games development. In *2018 IEEE 32nd International Conference on Advanced Information Networking and Applications (AINA)*. IEEE, Krakow, PL. <https://doi.org/10.1109/AINA.2018.00122>

A. Rollings and D. Morris. 2003. *Game Architecture and Design: A New Edition*. New Riders, Indianapolis, USA.

Anne Rozinat, RS Mans, Minseok Song, and Wil MP Van der Aalst. 2008. Discovering colored Petri nets from event logs. *International Journal on Software Tools for Technology Transfer* (2008). <https://doi.org/10.1007/s10009-007-0051-0>

Rudy Rucker. 2003. *Software engineering and computer games*. Addison-Wesley Harlow, Boston, USA.

Ubisoft. 2017. Tom Clancys Ghost Recon Wildlands, video game, developed and published by Ubisoft.

Wil Van Der Aalst and Kees Max Van Hee. 2004. *Workflow management: models, methods, and systems*. MIT press, Cambridge, MA.

Wil MP van der Aalst. 1996. Structural characterizations of sound workflow nets. *Computing Science Reports* (1996).

Wil MP van der Aalst. 1997. Verification of workflow nets. In *International Conference on Application and Theory of Petri Nets*. Springer Berlin Heidelberg, Berlin, Heidelberg. <https://doi.org/10.1007/3-540-63139-9_48>

Wil MP van der Aalst. 1998. The application of Petri nets to workflow management. *Journal of circuits, systems, and computers* (1998).

Wil MP van der Aalst and Christian Stahl. 2011. *Modeling business processes: a petri net-oriented approach*. MIT press, Cambridge, MA. <https://doi.org/10.7551/mitpress/8811.001.0001>

Wil M.P. van der Aalst and Arthur H.M. ter Hofstede. 2000. Verification Of Workflow Task Structures: A Petri-net-baset Approach. *Information Systems* (2000).

José Pablo Zagal, Miguel Nussbaum, and Ricardo Rosas. 2000. A model to support the design of multiplayer games. *Presence: Teleoperators & Virtual Environments* (2000). <https://doi.org/10.1162/105474600566943>