Laiza Costa Camurugy

Modelagem de jogos de *adventure* através de Machinations

Salvador

2017

Laiza Costa Camurugy

Modelagem de jogos de *adventure* através de Machinations

Monografia apresentada ao Curso de graduação em Ciência da Computação, Departamento de Ciência da Computação, Instituto de Matemática e Estatística, Universidade Federal da Bahia, como requisito parcial para obtenção do grau de Bacharel em Ciência da Computação.

Universidade Federal da Bahia – UFBA

Instituto de Matemática e Estatística

Departamento de Ciência da Computação

Orientador: Rodrigo Rocha Gomes e Souza

Salvador

2017

Laiza Costa Camurugy
Modelagem de jogos de *adventure* através de Machinations/ Laiza Costa Camurugy. – Salvador, 2017-
67 p. : il. (algumas color.) ; 30 cm.

Orientador: Rodrigo Rocha Gomes e Souza

Monografia – Universidade Federal da Bahia – UFBA
Instituto de Matemática e Estatística
Departamento de Ciência da Computação, 2017.

1. design de jogos. 2. diagramas. 3. desenvolvimento de jogos. 4. ferramenta. 5. jogos de adventure. I. Rodrigo Rocha Gomes e Souza. II. Universidade Federal da Bahia. III. Instituto de Matemática e Estatística. IV. Departamento de Ciência da Computação V. Modelagem de jogos de *adventure* através de Machinations

Laiza Costa Camurugy

Modelagem de jogos de *adventure* através de Machinations

Monografia apresentada ao Curso de graduação em Ciência da Computação, Departamento de Ciência da Computação, Instituto de Matemática e Estatística, Universidade Federal da Bahia, como requisito parcial para obtenção do grau de Bacharel em Ciência da Computação.

Trabalho aprovado. Salvador, 7 de abril de 2017:

Rodrigo Rocha Gomes e Souza
Supervisor

Professor
Christina Von Flach Garcia Chavez

Professor
Filipe Tiago Lima Pereira

Salvador
2017

Agradecimentos

Ao meu orientador Rodrigo Rocha Gomes e Souza por ter sido sempre presente em sua orientação me dando forças para que enfim pudesse concluir esta etapa, minha família, meus amigos e meu namorado que sempre me apoiaram e ofereceram suporte das diversas maneiras possíveis. Sem vocês eu não teria chegado onde estou hoje, então esse sucesso também é de vocês! A Dra. Lynn Alves, coordenadora do grupo de pesquisa Comunidades Virtuais que participei de 2009 a 2016, pela oportunidade de crescimento pessoal e profissional proporcionada e a todas as pessoas que trabalhei conjuntamente neste grupo! Aos meus colegas de curso, com os quais pude compartilhar conhecimentos e experiências ao longo destes anos, e por fim agradeço aos meus professores, por todo o conhecimento transmitido e por terem escolhido essa bela profissão que é a de ensinar.

Abstract

The development of video games also fits in the area of software development; this creates a need to use models and tools to design narratives and mechanics, to document, test and evaluate them during their creative process. This work presents a possibility of using the visual language and framework Machinations in the modeling of adventure games, creating patterns to represent elements of games. For this, we make a comparative analysis between a model made exclusively for this work and another related work. As a result we conclude that it is feasible to model adventure games on Machinations even though this genre is not the focus of the framework, and that there is a lack of good tools for game models and academic research in the area.

Keywords: game design, diagram, game development, tool, adventure games.

Resumo

O desenvolvimento de jogos também se enquadra como desenvolvimento de software; sendo assim cria-se a necessidade do uso de modelos e ferramentas para projetar narrativas e mecânicas, documentar, testar e avaliá-las durante seu processo criativo. Neste trabalho avaliamos a viabilidade de uso da linguagem visual e *framework* Machinations na modelagem de jogos de *adventure*, criando padrões para representar elementos desses jogos. Para isso, fazemos uma análise comparativa entre uma modelagem feita exclusivamente para este trabalho e um outro trabalho relacionado. Como resultado concluímos que é factível a modelagem de jogos de *adventure* no Machinations mesmo que este gênero não seja o foco do *framework*, e que existe uma carência de boas ferramentas para modelagens de jogos e de pesquisa acadêmica na área.

Palavras-chave: *design* de jogos, diagramas, desenvolvimento de jogos, ferramenta, jogos de *adventure*.

Lista de Figuras

Lista de Tabelas

Lista de abreviaturas e siglas

FPS	First Person Shooter
IF	Interactive Fiction
UML	Unified Modelling Language
MM	Micro-Machinations
MM AiR	MM Analysis in Rascal
MMORPG	Massively Multiplayer Online Role-Playing Game
MOBA	Multiplayer Online Battle Arena
PROMELA	Process or Protocol Meta Language

Conteúdo

1 Introdução

De acordo com o relatório "*2016 Global Games Market Report*" (Relatório do Mercado Global de Jogos) realizado pela NewZoo[1], gerou-se um total de 99.6 bilhões de dólares em receita no mercado mundial de jogos digitais, e a expectativa é que esse valor cresça para 118.6 bilhões até 2019. Com base nesses fatos e como jogos se enquadram como sistemas, não é surpresa que a área de desenvolvimento de jogos seja de interesse em Engenharia de Software e Ciência da Computação em geral.

Um jogo é um tipo de atividade lúdica, conduzida no contexto de uma realidade fingida, na qual os participantes tentam alcançar pelo menos um objetivo arbitrário e não trivial, agindo de acordo com as regras (ADAMS, 2009). Em (KASURINEN, 2016) foi conduzida uma pesquisa para entender as diferenças entre o tradicional desenvolvimento de software e o desenvolvimento de jogos digitais. Como o desenvolvimento de jogos tem um alto índice de imprevisibilidades e constantes mudanças durante sua concepção e desenvolvimento, métodos ágeis são comumente aplicados. Nessa mesma pesquisa, as diferenças de desenvolvimento percebidas são causadas pelos diferentes público-alvo, diferentes fontes de informação e diferentes usos dos produtos. Os jogos são produtos de entretenimento, enquanto sistemas de software tradicionais têm alguma atividade concreta ou problema a ser resolvido. O foco principal de um jogo não é o sistema e sim a experiência sensorial com a qual o jogador terá contato. Essa experiência é influenciada pela narrativa, pela arte, pelos efeitos sonoros e músicas, e pelas mecânicas do jogo.

Mecânicas de jogo são as regras, processos e dados de um jogo, definindo como o jogo progride, o que acontece e quando, e quais condições determinam a vitória ou a derrota (ADAMS; DORMANS, 2012). Elas são projetadas por *game designers* ou projetistas do jogo que têm como função criar e projetar aspectos que caracterizam o jogo. Antes que os *game designers* possam passar as definições de mecânicas projetadas para serem implementadas, é necessário ter certeza que essas mecânicas são viáveis, divertidas e funcionais, criando a necessidade de testes. Em (KASURINEN; STRANDÉN; SMOLANDER, 2013) os autores pesquisaram as ferramentas de *design* e desenvolvimento usadas por organizações de desenvolvimento de jogos. Dentre os resultados, temos que as ferramentas de software aplicadas também são uma parte importante do *design* do produto, pois a prototipagem é uma abordagem de projeto comum.

Os modelos que às vezes são usados para representar mecânicas de jogos, tais como o próprio código-fonte, máquinas de estados finitos, ou redes de Petri são complexos e não são realmente acessíveis para *designers* (ADAMS; DORMANS, 2012). Pesquisas

[1] Relatório disponível em: http://resources.newzoo.com/global-games-market-report

acadêmicas na área de desenvolvimento de ferramentas para o auxílio de *game design* são escassas. Nenhuma das ferramentas já desenvolvidas pode se considerar bem sucedida e aceita pela comunidade de desenvolvimento de jogos. Para além das linguagens visuais existentes para modelar mecânicas em forma de diagramas, existe a linguagem e *framework* Machinations, desenvolvida por Joris Dormans (DORMANS, 2009), que será o nosso objeto de estudo. O Machinations foi proposto para, além de ser uma linguagem visual de fácil compreensão para não-programadores, possibilitar executar o diagrama e validar mecânicas em tempo real através de uma ferramenta criada e disponibilizada online[2]. Esta linguagem foi escolhida tanto pela sua utilidade, como por ser relativamente nova. Não existem muitos documentos falando sobre a linguagem ou diagramas de exemplo disponibilizados na literatura.

O Machinations foi concebido com foco principal em simular e testar a economia interna de um jogo. A economia interna de um jogo é um sistema no qual seus elementos são produzidos, consumidos e trocados em quantidades quantificáveis. Esses elementos tangíveis e intangíveis são chamados de recursos, que vão desde moedas de troca a pontos de vida ou de experiência do jogador. A maioria dos jogos tem uma economia interna, embora a complexidade e a importância da economia interna varie consideravelmente de gênero para gênero (ADAMS, 2009). Mas de acordo com (ADAMS; DORMANS, 2012), jogos de *adventure* tipicamente não possuem economia interna, apenas mecanismos de *lock-and-key*[3]. Com isso surgiram questionamentos que motivaram este trabalho: seria viável modelar jogos de *adventure* com o Machinations, e em caso afirmativo, é possível criar um guia para modelar jogos de *adventure* de forma geral?

1.1 Objetivo do trabalho

O principal objetivo deste trabalho é propor e avaliar uma forma de modelar jogos de *adventure* no Machinations, além de difundir conhecimento sobre uma nova linguagem visual na área de desenvolvimento de jogos, e de possibilitar que pessoas fora da área de Ciência da Computação compreendam e possam modelar mecânicas antes de partirem para suas implementações.

1.2 Metodologia

Para atingir o objetivo do trabalho, realizamos as seguintes atividades:

- criação de guia para modelagem de jogos de *adventure* no Machinations;

[2] Ferramenta online disponibilizada: http://www.jorisdormans.nl/machinations/

[3] Mecanismos de *lock-and-key* são utilizados para controlar o progresso do jogador, bloqueando certas áreas até que o jogador realize tarefas ou consiga itens específicos para desbloqueá-las

- utilização do guia proposto para modelagem de um exemplo com Machinations;
- modelagem de uma fase do jogo de *adventure* Búzios: Ecos da Liberdade, já publicado e validado;
- modelagem do Cloak of Darkness, jogo de *adventure* textual amplamente conhecido;
- comparação dos modelos do Cloak of Darkness criados com Machinations e com redes de Petri (SOUZA, 2008).

1.3 Estrutura da monografia

No Capítulo 2, vemos conceitos relacionados a jogos, *game design* e jogos de *adventure* e suas características. Em adição, fazemos um breve estudo sobre os trabalhos relacionados ao nosso objeto de estudo. No Capítulo 3, explicamos a maioria dos elementos do Machinations, pois são necessários no Capítulo 4, onde propomos padrões para modelar jogos de *adventure* no Machinations. No Capítulo 5 modelamos o jogo de *adventure* Búzios: Ecos da Liberdade, publicado e validado, e o jogo Cloak of Darkness. Apresentamos uma avaliação comparativa com outro modelo baseado em rede de Petri de um trabalho relacionado. Por fim, no Capítulo 6 apresentamos a conclusão deste trabalho, suas limitações e trabalhos futuros.

2 Fundamentação teórica

2.1 Jogos de *adventure*

Jogos de *adventure* são aqueles em que o jogador assume o papel de protagonista em uma história interativa, incentivando o raciocínio lógico por meio da solução de *puzzles* ou quebra-cabeças, e incentivam a exploração do cenário, o qual possui objetos específicos os quais o jogador pode ou precisa interagir (CAMURUGY et al., 2010). Outra definição apresentada por (ADAMS, 2009), diz que a narrativa e a exploração são elementos essenciais de um jogo de *adventure*. Os desafios de combate e gestão econômica são reduzidos ou inexistentes.

2.1.1 Breve história

Jogos de *adventure* foram bastante populares nos primórdios da indústria de jogos eletrônicos. Computadores antigos possuíam pouca capacidade de memória e processamento, portanto os primeiros jogos foram puramente baseados em texto. Em jogos de *adventure* baseados em texto, comumente conhecidos como *Interactive Fictions* (IFs) ou traduzidos como jogos de Ficção Interativa, o jogador controla o personagem principal (avatar) através de entrada textual formando comandos de jogo, e o estado de jogo atual ou resultante é relatado por saída textual. O jogo é baseado em turnos, e geralmente envolve a exploração de uma série interconectada de quartos, onde o avatar pode examinar, mover e manipular objetos de jogo (VERBRUGGE; ZHANG, 2010). Colossal Cave Adventure (William Crowther and Don Woods, 1976) é amplamente considerado como o primeiro jogo de *adventure* baseado em texto. Um grupo de estudantes do MIT desenvolveu o jogo Zork (Infocom, 1980), também baseado em texto, o qual fez grande sucesso. Quando os computadores pessoais começaram a desenvolver capacidade gráfica, jogos de *adventure* ganharam ainda mais popularidade. Dentre os principais títulos da época, destacamos alguns como King's Quest (Sierra On-Line, 1984), The Secret of the Monkey Island (Lucasfilm Games, 1990) e Myst (Cyan, 1993), apresentados na Figura 1.

Com o passar dos anos a capacidade inovativa dentro do escopo de *adventures* foi diminuindo, e após o surgimento de outros gêneros de jogos como os *first-person shooters* (FPS), houve um declínio de popularidade de jogos de *adventure*. Apesar deste declínio, eles ainda continuam sendo lançados, alguns com sucesso de repercussão e vendas, como Machinarium (Amanita Design, 2009), The Walking Dead Series (Telltale Games, 2012), Gone Home (The Fullbright Company, 2013), Tales From The Borderlands (Telltale Games, 2014), Life is Strange (Dontnod Entertainment, 2015), alguns destes representados na

Figura 1 – The Secret of the Monkey Island, King's Quest e Myst, respectivamente.

Figura 2.

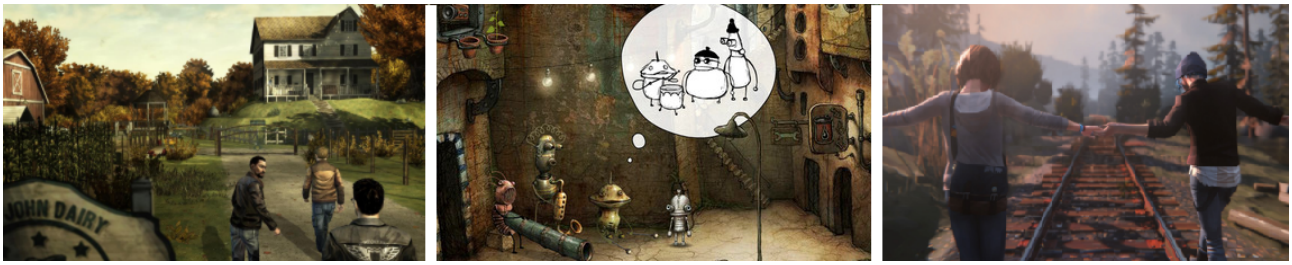


Figura 2 – The Walking Dead Episode 1, Machinarium e Life is Strange, respectivamente.

Os jogos de *adventure* escolhidos como objetos de estudo deste trabalho são o Búzios: Ecos da Liberdade (Comunidades Virtuais, 2011) e o Cloak of Darkness. O Búzios foi desenvolvido pelo grupo de pesquisa Comunidades Virtuais em 2010, e é um jogo de *adventure* sobre a Revolta dos Búzios que aconteceu na Bahia no fim do século XVIII. É um jogo de *point-and-click*, 2D e desenvolvido em Flash. O Cloak of Darkness é um IF criado por Roger Firth[1], desenvolvido para comparar sistemas de criação de IFs, e é considerado como o *"Hello world"* utilizado por programadores iniciando seu aprendizado em qualquer linguagem de programação.

2.1.2 Características

Apesar de existirem variações de implementação de qualquer gênero de jogo, existem características em comum as quais fazem com que um determinado jogo se encaixe em sua categoria. Dessa forma, descrevemos o escopo de jogos de *adventure* com base nas características mais comuns.

Um jogador controla um personagem principal (***avatar***) que realizará ações especificadas durante o jogo com o objetivo de conseguir chegar na condição de vitória, comumente chamada de zeramento do jogo. No jogo, temos **itens**, **objetos interativos**, personagens não-jogáveis (*Non-Player Characters* ou **NPCs**) e **cenários**. Os itens são

[1] Mais sobre: http://www.firthworks.com/roger/cloak/

objetos que podem ser coletados ou recebidos pelo jogador. Estes itens são armazenados em um **inventário**. A maioria dos jogos de *adventure* possui inventário, mas não todos[2].

NPCs são personagens sobre os quais o jogador não possui controle. Eles podem ser aliados amigáveis, neutros ou entidades hostis. NPCs são adicionados para fornecer recursos, ajudar o avatar na aquisição de habilidades especiais, dar conselhos, ou servir como um treinador para ajudar o jogador a desenvolver habilidades de resolução de problemas (JU; WAGNER, 1997).

Os cenários são ambientes pelos quais o jogador passa durante o percorrer do jogo. Consideramos objetos interativos como objetos com os quais o jogador pode realizar alguma interação. As interações podem ser acender uma lâmpada, abrir uma porta, empurrar um objeto, e assim por diante. Essas interações podem mudar o estado de um cenário, de um NPC e até de objetos e itens. Esses estados e seus valores são caracterizados por meio de **variáveis**. Nos exemplos de interações anteriores, a ação de abrir uma porta alteraria o valor da variável porta de 'aberta' para 'fechada', acender a lâmpada alteraria a variável de um ambiente de 'escuro' para 'iluminado', empurrar um objeto poderia ocasionar na queda de um outro objeto do cenário e mudar seu valor de variável para 'quebrado'.

Deixamos claro que os NPCs sempre estão situados em algum cenário, assim como itens coletáveis e objetos interativos. As ações realizadas por um jogador se enquadram em mover-se de um cenário para outro, coletar itens e interagir com NPCs e objetos do cenário. Além do uso de variáveis, a complexidade de um jogo de *adventure* pode ser aumentada com a adição de um sistema de *crafting*, ou seja, o jogador pode combinar itens para obter novos itens, bem como a adição de quebra-cabeças que deverão ser vencidos para continuar progredindo no jogo. Chamamos de estado final, aquele a partir do qual o jogador não pode realizar mais nenhuma ação. Este estado é classificado como um estado de vitória ou estado de derrota. Um dos pontos que se deve almejar em jogos de *adventure* é a possibilidade de sempre chegar em um estado final. Se um jogador pode jogar fora uma chave, por exemplo, e esta for a única chave necessária para prosseguir sua aventura, ele não terá mais como avançar e nem retroceder. Obviamente é uma situação que deve ser evitada, caso não o jogador teria que reiniciar o jogo. A Tabela 1 resume as características citadas acima.

2.2 Trabalhos relacionados

Em (TAYLOR; GRESTY; BASKETT, 2006) é demonstrado como diagramas UML (Unified Modelling Language) de caso de uso podem ser adaptados para projetar fluxos de jogos que não sejam apenas úteis para programadores, mas também para outros

[2] Loom é um exemplo de jogo de *adventure* que não possui inventário.

Item	Definição
Avatar	Personagem principal controlado pelo jogador
Cenário	Ambientes pelos quais o jogador passará ao longo do jogo
Itens	Podem ser obtidos em um cenário, recebidos por outro personagens, ou como recompensas ao completar objetivos
Inventário	Local onde os itens recebidos pelo jogador são armazenados
NPC	Personagem não jogável com o qual o jogador geralmente pode interagir
Objetos interativos	Objetos dos cenários com os quais o jogador pode interagir

Tabela 1 – Características comuns de jogos de *adventure*

profissionais que não sejam da área de computação, como roteiristas, *game designers* ou artistas.

O trabalho de Cook (COOK, 2007) concentrou-se na experiência do jogador e propôs diagramas de cadeias de habilidades (*skill chain diagrams*) para representar visualmente como os jogadores aprendem e adquirem habilidades.

Diversos autores utilizaram redes de Petri em representações narrativas devido à sua flexibilidade, bem como pela sua capacidade de modelar facilmente os recursos do jogo (VERBRUGGE; ZHANG, 2010). Em (ARAÚJO; ROQUE, 2009) os autores descrevem a abordagem com redes de Petri para modelar jogos, nos jogos "Europe 2045" (BROM; SISLER; HOLAN, 2007) e Karo (BALAS et al., 2008) elas foram utilizadas para modelagem na fase de especificação, e no trabalho de (SOUZA, 2008) temos uma modelagem e verificação de jogos de adventure através de redes de Petri.

3 Machinations

Neste capítulo apresentamos o *framework* Machinations, já que seus conceitos são essenciais no Capítulo 4 para modelarmos jogos de *adventure*. Machinations foi concebido por Joris Dormans para ajudar *game designers* a criar, representar e testar economias de jogos. Seus elementos são descritos nas seções seguintes, acompanhados de exemplos para ajudar no entendimento. Os conceitos passados neste capítulo são suficientes para a compreensão do trabalho desenvolvido, mas caso haja interesse em aprofundar tanto nas definições dos elementos do Machinations quanto em conceitos de *game design* sugerimos consultar o livro *Game Mechanics: Advanced Game Design* (ADAMS; DORMANS, 2012).

Os diagramas digitais de Machinations são dinâmicos e interativos, em contraste com esboços feitos em papel. Alguns exemplos modelados neste trabalho estão disponibilizados no link <https://github.com/fayalita/machinations_examples> para que o leitor possa executar, modificar ou usar como base para outras modelagens.

3.1 A ferramenta

Recomendamos o uso da ferramenta desenvolvida e disponibilizada online, para facilitar a validação e execução dos diagramas. É possível utilizá-la acessando através do link <http://www.jorisdormans.nl/machinations/>, ou fazendo o *download* do arquivo Machinations.swf por este mesmo link para poder utilizar remotamente. Na Figura 3 podemos observar:

- do lado esquerdo, um espaço com grade onde o diagrama será modelado;
- no canto superior esquerdo, o botão *Run* para iniciar a execução do diagrama;
- na parte superior direita, as abas que contém os elementos do Machinations, opções de importar ou salvar arquivos, opções de seleção ou cópia de elementos, etc;
- alguns campos de configurações do diagrama, como nome, autor, tamanho, etc.

3.2 Elementos básicos

Nesta seção, faremos uma breve descrição dos elementos mais básicos utilizados nos diagramas de Machinations. Os nomes utilizados serão escritos majoritariamente em português, mas sempre virão seguidos de seus nomes em inglês, visto que a ferramenta é toda em inglês.

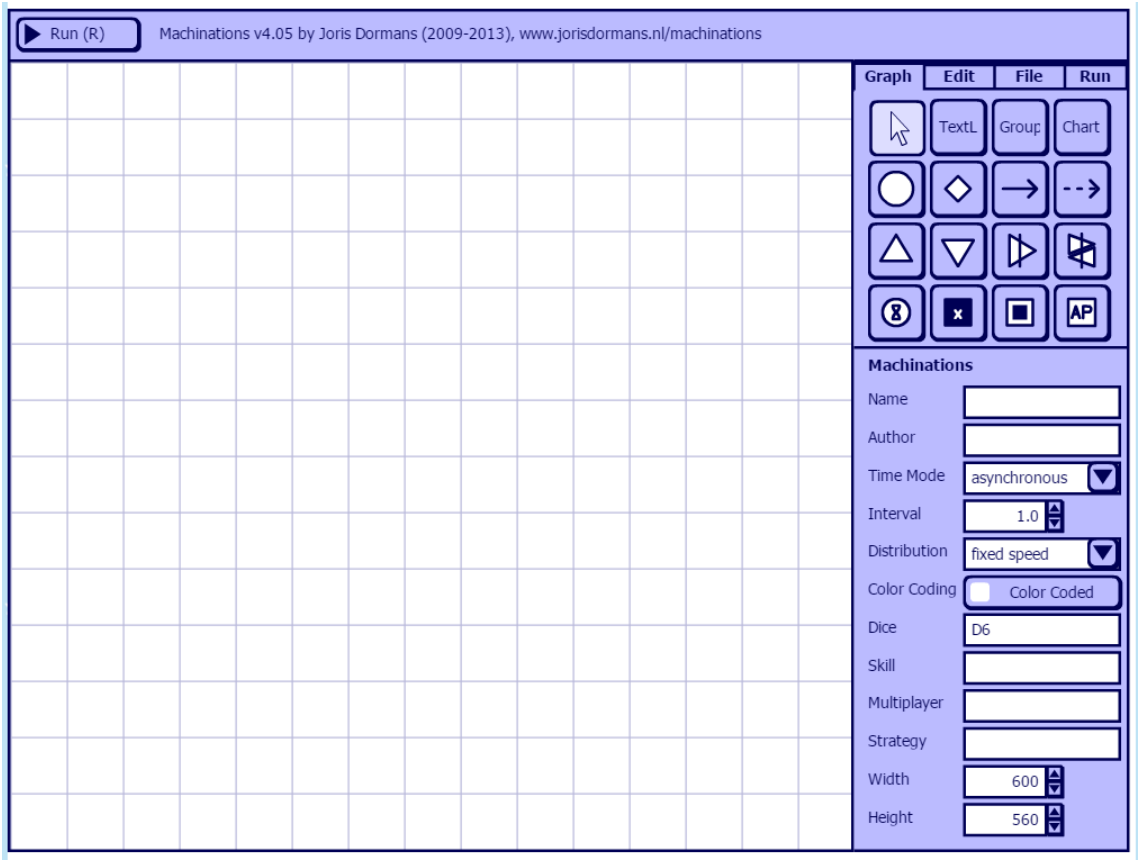


Figura 3 – Captura de tela da ferramenta

3.2.1 Reservatórios e recursos

Um recurso (*resource*) é um dos elementos básicos da economia de um jogo e refere-se a qualquer elemento que possa ser mensurado numericamente. Basicamente podemos considerar como um recurso algo que possa ser coletado, criado ou destruído. Exemplos tangíveis e intangíveis: dinheiro, pontos de vida, pontos de experiência, inimigos, itens, etc. Um reservatório (*pool*) em Ciência da Computação é geralmente um conjunto de recursos que são guardados prontos para uso. No Machinations, é um nó do tipo mais básico, onde os recursos se reúnem. Reservatórios e recursos são representados como na Figura 4. Vale ressaltar que todos os elementos do Machinations podem ter diferentes cores.

Figura 4 – Reservatórios.

3.2.2 Conexões entre nós

Para haver um fluxo de recursos entre os nós, existem conexões representadas por setas conectando os mesmos. Essas conexões possuem uma taxa de fluxo que define quantos recursos devem ser transmitidos. A taxa de fluxo de uma conexão é representada como um campo chamado rótulo (*label*), definido por constantes númericas e até fórmulas para definir resultados aleatórios. Não tendo nada neste campo, a taxa fica definida automaticamente como 1. Antes de exemplificar essas conexões, precisamos primeiramente falar sobre os **modos de ativação** de nós. Para que o diagrama seja executado, devemos definir como os nós se comportam e a partir de quais entradas (*inputs*) eles são acionados. Os modos de ativações dos nós podem ser de quatro tipos:

- automático, ou seja, a cada iteração ele é acionado;
- interativo, o qual depende da ação do jogador para ser acionado;
- nó de ação inicial, que dispara apenas uma vez, no início da execução;
- passivo, acionado apenas em resposta a um gatilho gerado por outros.

Em um diagrama, estes nós são representados como na Figura 5.

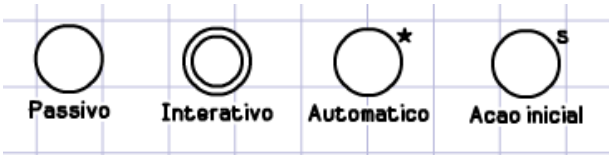


Figura 5 – Modos de ativação.

Quando um nó é acionado, o comportamento de resposta do nó é determinado pelo seu **modo de transmissão**. Este nó puxa recursos através de suas conexões de entrada, caso seu modo seja definido como *pull*, ou transmite recursos adiante através de suas conexões de saída, caso seu modo de transmissão seja definido como *push*. Na Figura 6 podemos notar os dois modos de transmissão definidos. Nós com modo *push* possuem o rótulo *p* para identificá-los, sendo omitido apenas quando não existem conexões de entrada.

Após uma única interação com os nós interativos da Figura 6, temos como resultado da transmissão de recursos apresentado na Figura 7.

Se a taxa de fluxo de uma conexão for X, mas os recursos a serem puxados ou empurrados através desta conexão forem Y, sendo Y $<$ X, então podem ocorrer duas situações:

- se o modo *push / pull* do nó for ***push / pull any***, então este nó irá transmitir ou puxar Y, mesmo Y sendo uma quantidade menor do que a especificada X;

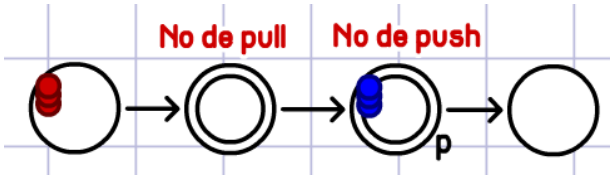


Figura 6 – Modos de transmissão de recursos.

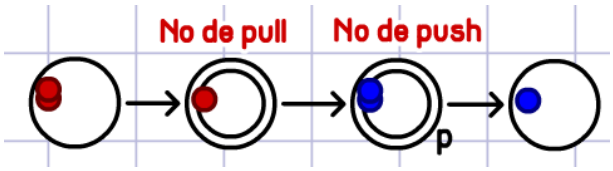


Figura 7 – Transmissão de recursos após acionamento de nós.

- se o modo *push / pull* do nó for ***push / pull all***, então este nó somente irá transmitir ou puxar recursos se existir a quantidade exata de recursos necessários.

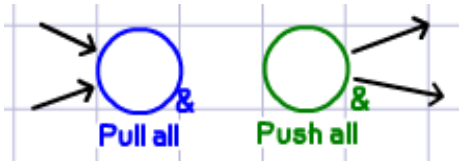


Figura 8 – Nós de *push* e *pull all*.

Note na Figura 8 que quando o nó é especificado como *push / pull all*, aparece o símbolo '&'. Já no modo *push / pull any*, que é o padrão, não há um símbolo específico.

Conexões também podem ter taxas de fluxo com intervalos. Esses intervalos são indicados utilizando a barra '/'. Por exemplo, uma conexão com a taxa de fluxo 2/6 significa que transmitirá 2 recursos a cada 6 passos de tempo. Na ferramenta do Machinations, cada iteração ou passo de tempo pode ter um tempo definido, em unidades de segundo, podendo ser um número fracionário. Esse tempo é determinado nas configurações gerais do diagrama no campo *"interval"*.

3.2.3 Mudanças de estado

O estado de um diagrama do Machinations é definido pela distribuição de recursos entre seus nós. Mudanças de estados podem ser utilizadas para modificar taxas de fluxo de conexões, ativar e disparar nós. Essas mudanças são possíveis através das chamadas conexões de estado. Conexões de estado são representadas por setas tracejadas, ligando o nó de origem a um alvo, que pode ser um nó ou uma conexão. As mudanças no alvo são

indicadas pelo rótulo da conexão. Elas podem ser de quatro tipos: modificadores de rótulo, modificadores de nó, gatilhos e ativadores.

3.2.3.1 Modificadores de rótulo

Modificadores de rótulo (*label modifiers*) conectam o nó de origem a um rótulo alvo, modificando seu valor em função da mudança de estado do nó de origem. Considere a seguinte fórmula:

$$L_{t+1} = L_t + MxΔS \tag{3.1}$$

L_t representa o rótulo alvo no tempo atual t, M representa o rótulo da conexão de estado e ΔS representa a variação do estado S do nó de origem. Tal modificação no rótulo L ocorre apenas no seguinte ciclo de tempo t + 1.

Caso o rótulo L seja alvo de múltiplos modificadores, a soma de todas as modificações devem ser somadas para encontrar o novo valor:

$$L_{t+1} = L_t + \sum(MxΔS) \tag{3.2}$$

Considere o exemplo da Figura 9. Antes de executarmos uma ação, o reservatório **A** contém 50 recursos, enquanto os reservatórios **B** e **C** não possuem nenhum. Se não houvesse a conexão de estado exemplificada, a cada iteração os reservatórios B e C receberiam um recurso cada. Observe que com o modificador de rótulo **+2** na conexão, haverá uma modificação no fluxo entre A e C de + 2 x ΔS que seria a variação de recursos no nó B.

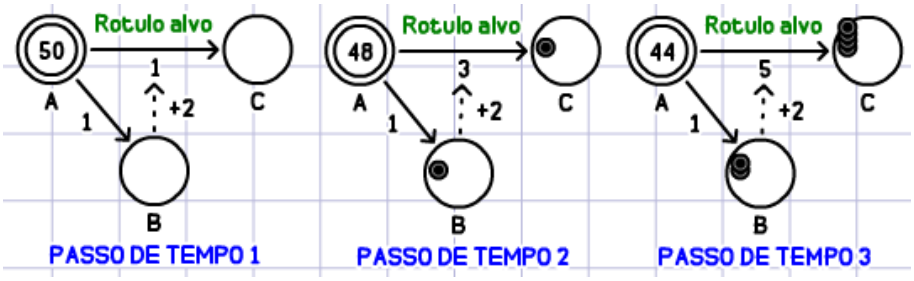


Figura 9 – Exemplo de um modificador de rótulos.

- Passo de tempo 1: $L_t = 1$.
- Passo de tempo 2: $L_{t+1} = L_t$ + 2 x ΔS = 1 + 2 x 1 = 3.
- Passo de tempo 3: $L_{t+1} = L_t$ + 2 x ΔS = 3 + 2 x 1 = 5.

Modificadores de rótulo também podem ser negativos, diminuindo a taxa de fluxo de um rótulo alvo.

3.2.3.2 Modificadores de nó

Diferentemente dos modificadores de rótulo, os modificadores de nós (*node modifiers*) conectam dois nós, mas possuem o mesmo princípio. Cada variação no nó de origem será multiplicada pelo rótulo da conexão modificadora e adicionada a quantidade de recursos existentes no nó alvo:

$$N_{t+1} = N_t + \sum(Mx\Delta S) \tag{3.3}$$

e para vários modificadores:

$$N_{t+1} = N_t + \sum(Mx\Delta S). \tag{3.4}$$

Observe na Figura 10 dois tipos de modificadores de nós, um positivo e outro negativo. Os nós iniciais A e B são acionados no início da execução do diagrama, transmitindo recursos para os nós que eles se conectam. Como o estado destes nós foi alterado, suas conexões passam a modificar o nó alvo com + 3 x ΔS recursos e - 2 x ΔS recursos, sendo ΔS a variação de estado nos nós de origem.

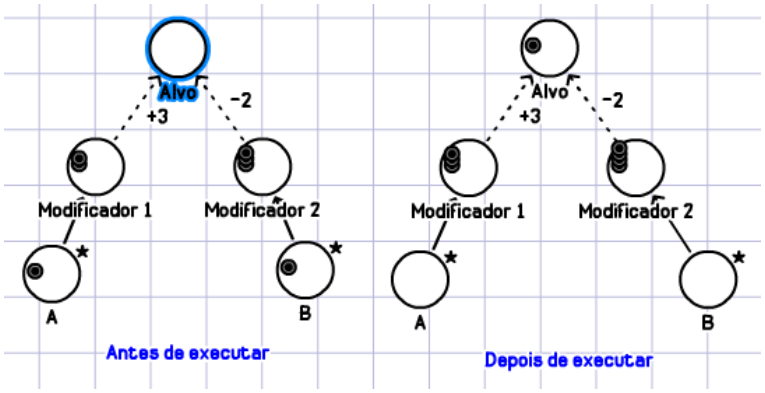


Figura 10 – Exemplo de modificadores de nó.

3.2.3.3 Gatilhos

Como o nome já diz, os modificadores gatilhos (*triggers*) acionam um determinado elemento do diagrama, seja ele um rótulo ou um nó, quando todas as condições de suas entradas forem atendidas. Se o seu alvo for um nó, ele irá puxar ou empurrar recursos de acordo com seu modo, e se o alvo for um rótulo ele irá transmitir a quantidade de recursos indicada.

No exemplo da figura 11 vemos que apenas quando **ambos** os reservatórios A e B transmitem seus recursos correspondentes resulta no acionamento do gatilho e transmissão de recurso de D para E.

[1] Recursos estão coloridos para melhor visualização

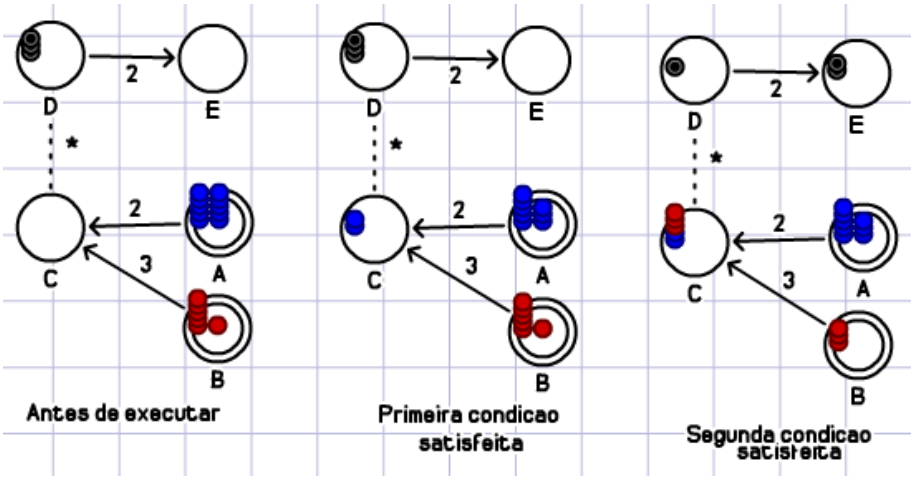


Figura 11 – Exemplo de gatilho aplicado a um nó.[1]

3.2.3.4 Ativadores

Ativadores (*activators*) conectam nós e podem ativar ou suspender o nó alvo dependendo da condição de seu(s) nó(s) de origem. Essas condições são descritas a partir de expressões aritméticas. Dentre as possibilidades estão:

- Condições de igualdade: == x, > x, >= x, <= x, < x, != x, sendo x $\in \mathbb{Z}$. Exemplo: > 2, <= 5, etc.
- Intervalo de valores: x-y. Exemplo: 4-10. [2]

Podemos ver no exemplo modelado na Figura 12 que para transportar o recurso madeira para o nó de inventário do jogador, são necessários lenhadores (também representado por recursos) no reservatório floresta. A cada 5 madeiras transportadas, um lenhador volta para seu reservatório de lenhadores. Quando não existe mais nenhum recurso de lenhador no reservatório de floresta, não há como movimentar mais as madeiras para o inventário do jogador.

Na tabela da Figura 13 temos um resumo dos tipos de rótulos de conexões, juntamente com o formato que devem ser utilizados e exemplos de uso.

3.3 Tipos avançados de nós

Para facilitar a visualização dos diagramas, (ADAMS; DORMANS, 2012) criou nós especializados. As funções desempenhadas por esses nós podem ser simuladas com nós básicos, mas o trabalho seria maior e deixaria mais complicado o entendimento.

[2] O intervalo é considerado fechado para o valor mínimo e para o máximo: [min, max]

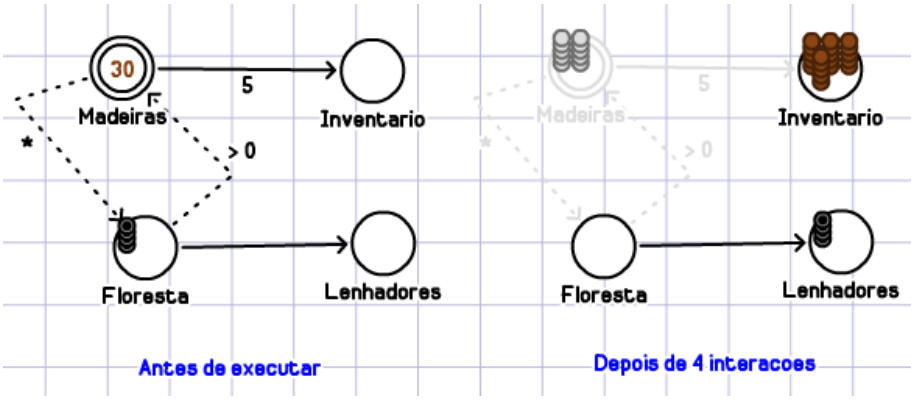


Figura 12 – Exemplo de um ativador.

Label Types	Format	Examples
Flow rate:	x	0; 2; 3; 0.5; 1.3
Random flow rate:	Dx; yDx; x%	D6; 2D5; D3-D2; 20%; 50%
Intervals:	x/y	1/4; 2/2; D6/3; D3/(D6+2)
Multipliers:	x*y	2*50%; 3*D3
All resources:	all	all
Draw randomly:	drawx	draw1; draw2; draw5

Figura 13 – Tabela resumindo tipos de rótulos de conexões (ADAMS; DORMANS, 2012).

3.3.1 Portões

Portões (*gates*) redistribuem recursos mas não os armazenam. Suas conexões de saída não possuem taxas de fluxo, e sim probabilidades (em forma de porcentagem ou números representando pesos) e condições (também descritas em expressões aritméticas). Vale ressaltar que todas as conexões de saída de um portão devem ser do mesmo tipo.

- Probabilidade com porcentagem: a probabilidade será igual a porcentagem indicada. A soma das probabilidades não deve ultrapassar 100%.
- Probabilidade com pesos: a chance de transmissão de um recurso por uma conexão será uma média aritmética simples dos pesos

$$\overline{p} = \frac{1}{n} \sum_{i=1}^{n} p_i \tag{3.5}$$

sendo n o número de conexões de saída e p_i o peso de determinada conexão.

Se a soma das probabilidades for menor que 100%, ou nenhuma das conexões de saída forem atendidas, os recursos excedentes serão destruídos. Em contraposição, se duas ou mais condições forem atendidas, o recurso será duplicado.

Todos os nós avançados descritos nessa seção possuem os mesmos modos de ativação que reservatórios, descritos na subseção 3.2.2, exceto por nós de estado final que são descritos em 3.3.6.

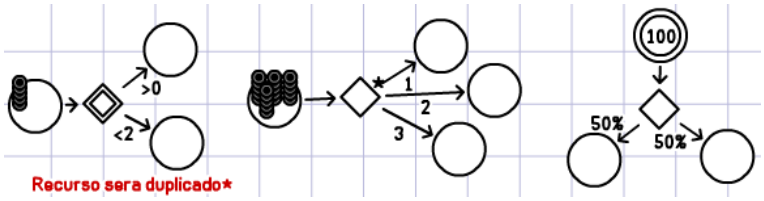


Figura 14 – Exemplo de portões.

Na Figura 14, o primeiro exemplo mostra um recurso que será duplicado, pois as duas condições serão atendidas, o segundo mostra um portão com probabilidades por peso e o terceiro um portão com probabilidades percentuais, havendo uma chance igual de 50% para ambas as conexões.

3.3.2 Fontes

Esse tipo de nó chamado de fonte (*source*) gera recursos. Eles não possuem conexões de entrada, e podem conter uma quantidade infinita de recursos.

Figura 15 – Representação de fontes.

3.3.3 Drenos

Como oposto de uma fonte, os drenos (*drains*) consomem ou destróem recursos. Os recursos consumidos são determinados pelas taxas de fluxo de suas conexões de entrada. Para consumir todos os recursos vindos de uma determinada conexão, basta colocar como rótulo da conexão a palavra *all*, que significa “todos”.

Na figura 16 vemos um exemplo de uso tanto de uma fonte, quanto de um dreno. A fonte é utilizada para gerar soldados, e a cada 5 soldados gerados são descontados 2 de dinheiro do jogador, caracterizando uma compra.

3.3.4 Conversor

Modelado para converter recursos, os conversores (*converters*) consomem os recursos de entrada e criam os recursos de saída. Portanto, podem ser modelados também com

Figura 16 – Exemplo de dreno e fonte.

drenos e fontes, como exemplificado na Figura 17.

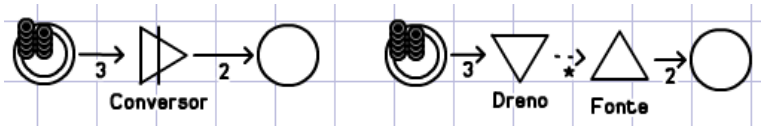


Figura 17 – Conversor e sua representação com dreno e fonte.

3.3.5 Trocador

Os trocadores (*traders*) são utilizados para o câmbio de recursos. Um número x de recursos é trocado por um número y de recursos, troca esta que só ocorre quando existe a quantidade necessária de ambos os recursos participantes da troca. Nós trocadores apenas alteram a posse de recursos, não necessariamente criando ou destruindo recursos como ocorre com conversores. Utilizando o exemplo de compra de soldados demonstrado na Figura 16, podemos modelar a mesma situação com um trocador de acordo com o diagrama na Figura 18.

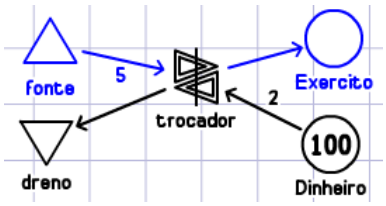


Figura 18 – Troca de recursos.

3.3.6 Condições de fim

Os jogos podem ser finitos, ou seja, têm um estado final (*end state*) seja de vitória ou derrota, ou podem ser infinitos, a exemplo jogos MMORPGs (*Massive Multiplayer Online Role-Playing Game*), jogos de simulação, jogos de mundo aberto, etc. Para determinarmos um estado final de um jogo no Machinations, utilizamos condições de fim para acionarmos o estado final e encerrar a execução do diagrama. Condições de fim são ativadas por ativadores.

Como representado na Figura 19, o jogador vence o jogo ao abrir a porta almejada. Essa porta está trancada, e a ação de abrir a porta só é liberada caso a chave necessária esteja no inventário. Tendo a chave obtida com o chaveiro por meio de troca por dinheiro, o jogador poderá acionar o nó de abrir a porta, tornando a condição de vitória verdadeira e terminando a execução do diagrama.

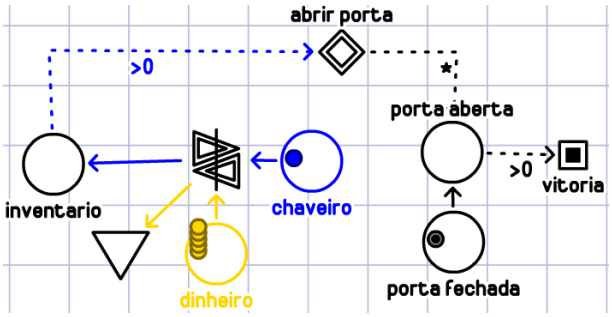


Figura 19 – Estados finais.

3.3.7 Registros

Registros (*registers*) são utilizados para facilitar a manipulação de dados numéricos. Quando marcados de tipo ativo, possuem um valor inicial determinado em sua configuração que pode ser modificado dinamicamente pelo jogador durante a execução do diagrama. Quando do tipo passivo, possuem um campo em que é possível colocar uma fórmula para calcular o seu valor. Essa fórmula é baseada pelos valores de entrada. Por padrão, conexões de estado que ligam dois registros recebem uma letra do alfabeto para representá-las.

Figura 20 – Registros.

Observe na Figura 20 que no primeiro caso, o registro é utilizado para mudar a taxa de fluxo de uma conexão, e no segundo caso dois registros do tipo ativo são usados para calcular o valor do terceiro registro, do tipo passivo. Esse terceiro registro é o resultado da soma dos outros dois. Além de expressões numéricas, podem ser utilizadas as palavras min e max representando os valores mínimo e máximo dentre as entradas. Vale também citar que podem ser utilizados parênteses nas expressões numéricas para indicar precedência de operadores.

3.3.8 Atrasos e filas

Nós de atraso (*delays*) são utilizados para aguardar uma determinada quantidade de passos de tempo para que os recursos possam fluir. Os nós de fila (*queues*) possuem o mesmo princípio, exceto que após esperar o tempo especificado, transmitem apenas um recurso por vez. A quantidade de passos de tempo que nós de atraso e nós de fila aguardam para transmissão de recursos é definida pelos rótulos de suas conexões de saída. Observe um exemplo básico na Figura 21, cujo nó de fila transmite um recurso a cada três passos de tempo.

Nós de atraso e nós de fila podem ser utilizados para criar efeitos temporizados. Segue um exemplo na Figura 22, onde foi modelado um sistema de uso de *skills* para um jogador de um jogo MOBA (*Multiplayer Online Battle Arena*). *Skills* são habilidades específicas dos personagens que o jogador controla. Alguns personagens para utilizarem determinadas *skills* precisam gastar uma determinada quantidade de *mana*, um recurso finito que possui regeneração. Cada *skill* possui um tempo de recarga, que foi modelada com o uso de nós de atraso. Para criar este efeito temporizado, conectamos os nós de atraso a nós de fonte com ativadores. Caso os nós de atraso possuam recursos em espera, suas respectivas fontes são ativadas, gerando um recurso a cada passo de tempo. Esses recursos gerados são utilizados para contabilizar o tempo decorrido. Os drenos destroem os recursos de tempo e de *mana* quando o tempo de recarga acaba.

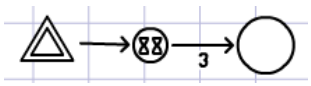


Figura 21 – Nó de fila que transmite 1 recurso a cada 3 passos de tempo.

3.3.9 Gatilhos reversos

Um gatilho reverso (*reverse trigger*) é uma conexão de estado representada com o sinal de exclamação '!' em seu rótulo. Caso seu nó de origem tente puxar a quantidade de recursos especificada pela soma dos rótulos de suas conexões de entrada mas não seja possível, ele acionará o gatilho reverso, acionando o nó em que sua saída está conectada. No exemplo da Figura 23, quando não existe mais dinheiro para ser drenado na compra de tropas, um gatilho reverso é acionado para pedir 5 recursos do nó banco.

3.3.10 Outros elementos

- *Color coding*: se essa opção não for marcada, as cores serão meramente ilustrativas. Caso sim, se uma conexão entre nós for de uma cor diferente da cor do nó de origem, então esta conexão transmitirá ou terá sua condição satisfeita levando em conta

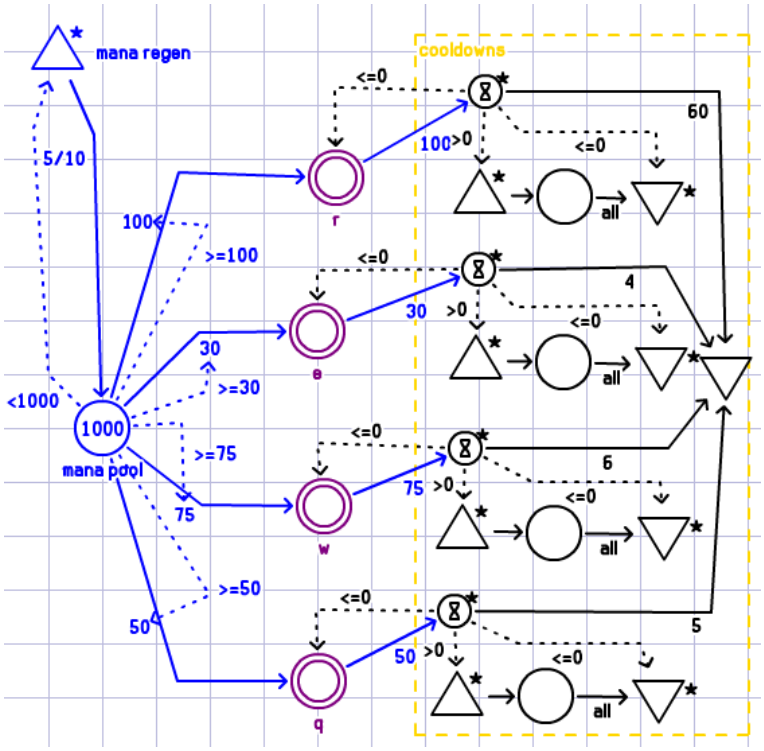


Figura 22 – Modelagem do uso de *skills* em um MOBA com o uso de um nó de atraso.

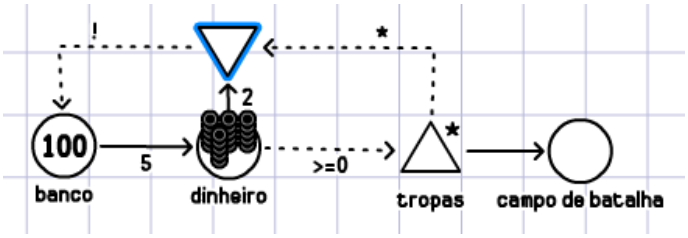


Figura 23 – Exemplo de um gatilho reverso.

apenas recursos de sua própria cor. Essa propriedade pode ser bem útil, visto que recursos com cores diferentes podem se agrupar em um mesmo reservatório. As possíveis cores utilizadas na ferramenta são: *Black, White, Red, DarkRed, Orange, OrangeRed, Yellow, Gold, Green, Lime, Blue, LightBlue, DarkBlue, Purple, Violet, Teal, Gray, DarkGray, e Brown*. Os nomes não são *case sensitive*, ou seja, não dependem das letras serem maiúsculas ou minúsculas.

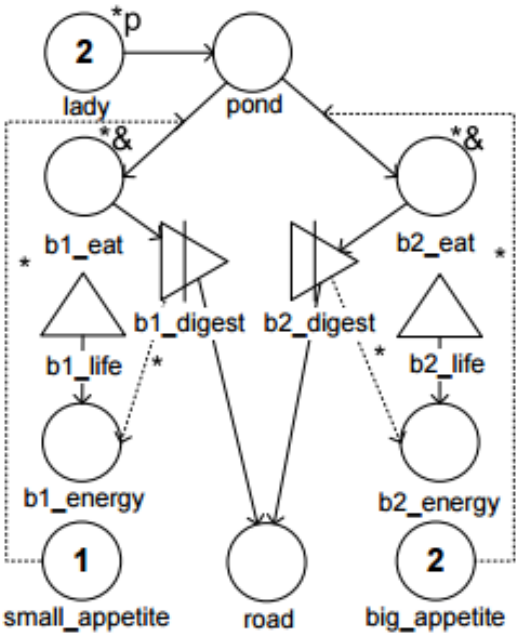
- *TextLabel*: texto meramente ilustrativo.
- *GroupBox*: caixa tracejada com título, meramente ilustrativa.
- *Chart*: gráfico utilizado para colher dados durante a execução do diagrama. Os dados são colhidos ao conectar nós com o gráfico. Ele pode armazenar dados de várias execuções, os quais podem ser exportados em um arquivo de formato CSV.

- ArtificialPlayer: permite a definição de um código para controlar outros nós do diagrama. É útil para a automatização de testes. Eles não são conectados a nada, e são representados por um quadrado com as siglas 'AP' em seu interior. No livro de (DORMANS, 2009) os possíveis comandos para criação do código de um jogador artificial são definidos, bem como há maiores explicações sobre múltiplas execuções e coleta de dados com gráficos.

3.4 Micro-Machinations

Micro-Machinations (MM) é um subconjunto estendido formalizado de Machinations desenvolvido por Paul Klint e Riemer van Rozen(KLINT; ROZEN, 2013) visando a prototipagem e validação de software, e permite não só a simulação, mas também a análise formal de projetos de jogos.

MM é uma DSL (Domain-Specific Language) que possui tanto um modelo visual, como uma linguagem textual que representa este modelo. Além do subconjunto retirado do Machinations, MM também adiciona novos recursos, especialmente a modularização. Seu propósito é ser uma linguagem de script que possa ser embutida em motores de jogo, permitindo a interação entre regras e elementos do jogo. A Figura 24 mostra um exemplo na forma visual e textual.




```
1  unit Bread      : "bread crumbs"
2  unit Droppings  : "bird residue"
3  unit Energy     : "bird energy"
4  pool BIG_APPETITE of Bread at 2
5  pool SMALL_APPETITE of Bread at 1
6  auto push all pool lady of Bread at 2
7  pool pond of Bread
8  pool road of Droppings
9  lady --> pond
10 auto all pool b1_eat of Bread
11 pond -SMALL_APPETITE-> b1_eat
12 auto converter b1_digest
13 from Bread to Droppings
14 b1_eat --> b1_digest
15 b1_digest --> road
16 pool b1_energy of Energy
17 source b1_life of Energy
18 b1_digest .*.> b1_energy
19 b1_life --> b1_energy
20 auto all pool b2_eat of Bread
21 pond -BIG_APPETITE-> b2_eat
22 auto converter b2_digest
23 from Bread to Droppings
24 b2_eat --> b2_digest
25 b2_digest --> road
26 pool b2_energy of Energy
27 source b2_life of Energy
28 b2_digest .*.> b2_energy
29 b2_life --> b2_energy
```

Figura 24 – Exemplo de modelo em Micro-Machinations(KLINT; ROZEN, 2013).

3.4.1 Análise formal

O trabalho sobre MM (KLINT; ROZEN, 2013) trouxe os seguintes questionamentos. Dado um estado de jogo, quais são os valores dos reservatórios, quais nós estão ativos e o que eles fazem? Dado um estado de jogo, quais são as possíveis transições? Existe algum fluxo que nunca acontece? Um estado desejado é sempre acessível, por exemplo, pode o jogo ser ganhado?

Para que essas perguntas possam ser respondidas, é necessária uma análise formal do modelo. MM traz uma representação textual que pode ser facilmente analisada. A partir dela, é utilizado o *framework* MM Analysis in Rascal (MM AiR) desenvolvido para analisar e simular modelos MM usando a linguagem de metaprogramação Rascal[3] e o verificador de modelos Spin[4]. Este *framework* possui as seguintes funcionalidades:

- verificar restrições contextuais - traz *feedback* quando os modelos estão incorretos ou não passam análise contextual;
- simular modelos MM - fornece visualização gráfica de um modelo e permite aos usuários inspecionar estados, escolher transições e navegar pelo modelo avançando e retrocedendo;
- traduzir MM para PROMELA - realiza a tradução de modelos MM para a linguagem PROMELA[5] (*Process or Protocol Meta Language*), sendo esta uma linguagem de entrada para o verificador de modelos Spin;
- verificar modelos MM em Spin - permite a análise de acessibilidade de forma escalável;
- analisar acessibilidade - reporta mensagens com indicações como nós que nunca puxam ou empurram recursos, conexões que nunca transmitem recursos ou gatilhos que nunca são ativados;
- repetir comportamentos - disponibiliza a opção de repetir simulações de forma guiada.

Um aspecto importante de análise é o da acessibilidade (*reachability*). Dado um estado inicial s e um estado alvo t, é possível chegar no estado t a partir do estado s? Fazer esse tipo de análise é difícil pois requer o cálculo de todos os caminhos possíveis através do grafo do jogo. Normalmente, não podemos calcular todas as execuções possíveis de programas devido ao grande número de possibilidades (KLINT; ROZEN, 2013). O MM reduziu não-determinismos por conta da explosão de estados que podem causar. Os

[3] http://www.rascal-mpl.org/
[4] http://spinroot.com/spin/whatispin.html
[5] Referência rápida:http://spinroot.com/spin/Man/Quick.html

nós passam a ter prioridades, pois foi identificado que a origem de não-determinismos se deve a competição de nós por recursos e o uso do modificador *any*. Cada nó pode atuar uma vez durante um passo de tempo. Essa ordem não é definida, portanto este aspecto também pode resultar em não-determinismo. Dessa forma, em (KLINT; ROZEN, 2013) é especificado que nós e seus modificadores são processados na seguinte ordem: *pull all*, *pull any*, *push all*, *push any*.

3.5 Micro-Machinations Library

A Micro-Machinations Library (MM Lib) é uma biblioteca de software escrita em C ++ para incorporar MM em jogos e ferramentas[6] (ROZEN; DORMANS, 2014). Nesse trabalho foi demonstrado o estudo de caso AdapTower, jogo implementado em C# e que incorpora a MM Lib. Na Figura 25 podemos observar uma captura de tela de seu protótipo, e através do link <https://www.youtube.com/watch?v=YzsKaJEX4D4&t=2s> podemos visualizar um vídeo de seu funcionamento.

Figura 25 – Captura de tela do AdapTower(ROZEN; DORMANS, 2014).

[6] Código fonte: https://github.com/vrozen/MM-Lib

4 Jogos de adventure no Machinations

Tendo visto uma breve introdução sobre as características básicas de jogos de *adventure* no Capítulo 2 e os conceitos básicos utilizados no Machinations no Capítulo 3, podemos agora propor uma abordagem na modelagem de jogos de *adventure* utilizando este *framework*. Devemos aqui enfatizar que esta não será necessariamente a única solução existente, mas que pode haver outras soluções possíveis. Vamos começar pelos conceitos básicos e aos poucos poderemos aumentar a complexidade.

4.1 Estruturas

4.1.1 Jogador e cenários

Esta subseção traz conceitos caso o jogo a ser modelado envolva cenários diferentes e movimentação do jogador entre eles. Se o jogo a ser modelado se passa em um único cenário, com foco nas interações com objetos e NPCs que nele se encontram, apenas os conceitos das próximas subseções serão utilizados.

O personagem principal deverá sempre estar em um cenário. Portanto, consideraremos-no um recurso que transitará entre nós. Como iremos utilizar codificação por cor, vamos definir uma cor única para o recurso que representa o jogador, sendo esta definida como a cor preta nos exemplos seguintes.

Segundo (DORMANS, 2009) o *framework* não foi desenvolvido para explorar o *level design*[1] de jogos em muitos detalhes, funcionando melhor com representações simples de espaços de jogo, utilizando reservatórios para representar localizações. Cenários são ambientes que representam localizações possíveis dentro do jogo, então os representaremos como nós do tipo reservatório.

Na Figura 26 existem duas soluções representando a movimentação de um jogador entre quatro cenários neste caso representados por salas. A sala 2 tem acesso às outras três salas, e as salas 1, 3 e 4 tem apenas um acesso para a sala 2. A primeira solução é recomendada pela sua simplicidade, resultando neste exemplo um diagrama com apenas 4 nós. A segunda solução é apenas recomendada quando é indispensável a informação de qual direção o jogador está se movimentando. Para isso é preciso que as direções sejam bem definidas, como Norte, Sul, Leste e Oeste. Neste caso, cada cenário terá seus respectivos nós de portão indicando as saídas nas possíveis direções, sendo estas saídas conexões de

[1] O *level design* é responsável por aplicar todas os componentes definidos pelo game design para a criação da experiência que o jogador terá durante o jogo. Elaborando elementos como o espaço onde o jogo ocorre, os desafios enfrentados pelo jogador, o ritmo dos eventos e a atmosfera do mundo do jogo.

recursos que se ligam a um outro cenário.

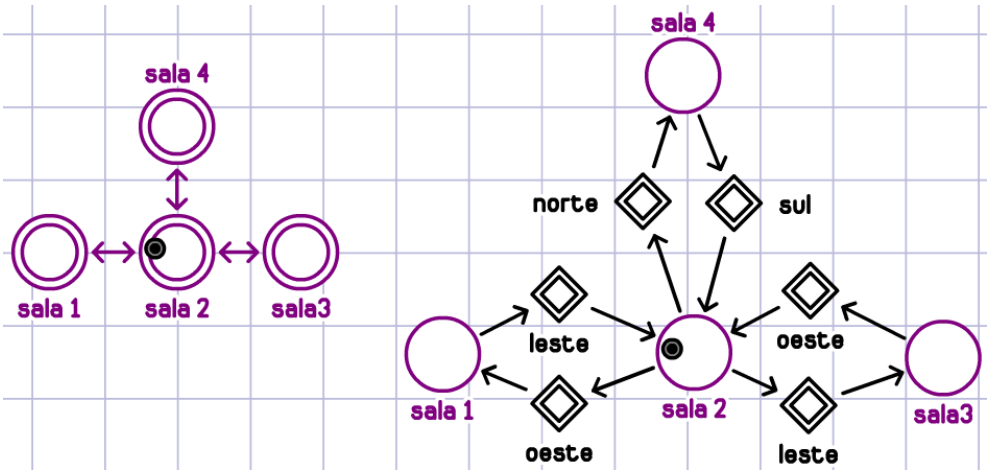


Figura 26 – Representação do jogador e de cenários.

4.1.2 Itens coletáveis

Os itens coletáveis serão modelados como recursos. Eles podem ser uma instância única, como uma chave que estará em cima de uma mesa de um cenário, ou de várias instâncias, como um pedaço de madeira que estará em um reservatório de floresta. Após coletados, vão para o inventário do jogador, modelado como um reservatório. Diferenciaremos os itens com cores diferentes, portanto a opção de *Color Coding* deve ser marcada. Na representação da Figura 27, foram acrescentados dois elementos *TextLabel* com o nome dos itens para efeito de ilustração. Em um diagrama isso não é recomendado pois o *TextLabel* não acompanha a posição do recurso caso ele seja transmitido para outro nó.

Figura 27 – Representação de itens coletáveis.

Em jogos de adventure, para que o jogador colete um item ele deve estar no mesmo cenário que este item. Neste caso, propomos o seguinte padrão para a coleta de itens:

- um nó de portão será designado para a ação de coletar;
- entre o reservatório do cenário e o nó de portão de coleta haverá uma conexão de estado com condição de que exista um recurso de jogador neste reservatório;
- para cada item de cor distinta no cenário, deverá ser criada uma conexão de recurso com sua respectiva cor e taxa de fluxo, também entre o nó de portão e o cenário;

- caso existam outras condições para a coleta de alguns desses itens, outras conexões de estado também serão criadas para habilitar ou não a coleta.

Na Figura 28 o jogador ainda não chegou no nó da sala de tesouros onde os itens coletáveis se encontram, portanto a coleta dos mesmos ainda não está disponível. Após chegar na sala de tesouros, exemplificado na Figura 29, a coleta de itens é habilitada.

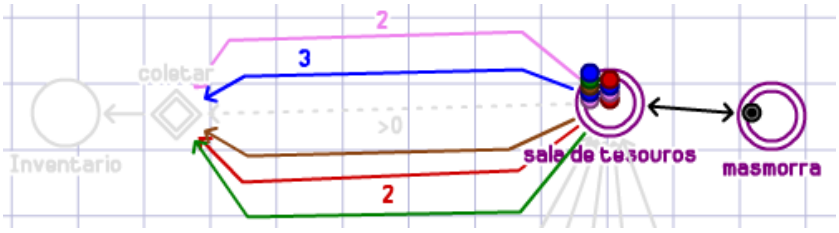


Figura 28 – Jogador antes de alcançar a sala dos itens que serão coletados.

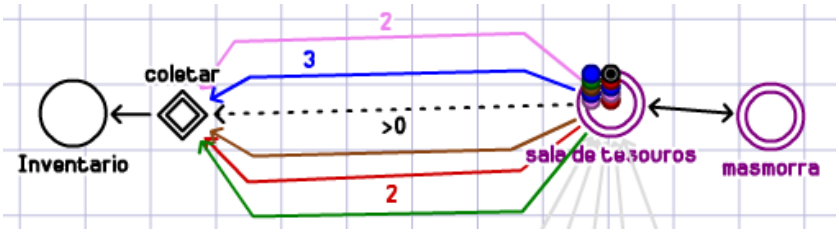


Figura 29 – Coleta de itens habilitada após jogador alcançar a sala requerida.

4.1.3 Variáveis e interações com objetos

Ações que envolvam variáveis do sistema ou condições para que uma ação seja tomada são modeladas com nós de portões. Estes nós não armazenam recursos e sim transmitem, assim podemos fazer conexões de estado com eles para especificar condições de ativação, e conexões de recursos para transmissão de recursos caso as condições tenham sido satisfeitas. Na Subseção 4.1.2 utilizamos um nó de portão para representar a ação de coleta de itens. Na Figura 30 observamos que o nó que representa a ação de "abrir a porta" é representado por um nó de portão, cuja ativação apenas ocorre quando a condição de ter uma chave marrom no inventário é atendida.

4.1.4 Interações com NPCs

Caso os NPCs a serem modelados não se movam entre cenários e tenham interações específicas com eles, sua modelagem será a partir de nós de reservatório. Neles serão armazenados os recursos referentes ao seu diálogo, a itens que estejam em sua posse ou alguma outra propriedade que precise ser representada. Além disso, estarão ligados a nós

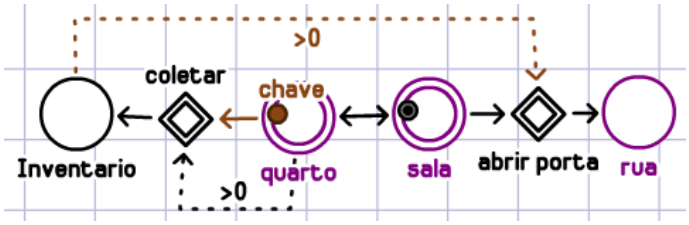


Figura 30 – Representação de interações com objetos.

de portão que servirão para a ação de fala. Estes nós puxarão seus respectivos recursos através de um nó fonte. A forma que o recurso de diálogo é tratado no reservatório do NPC pode variar de acordo com o *design* do jogo. Para fins de deixar o diagrama mais limpo visualmente, podem ser acrescentados drenos para destruir os recursos de diálogo em situações que o jogador fala com um NPC e a fala deste permanece a mesma após interações consecutivas. Para facilitar e diminuir a quantidade de nós em um diagrama, pode-se inserir uma única fonte que transmitirá os recursos de diálogo para os reservatórios de NPCs.

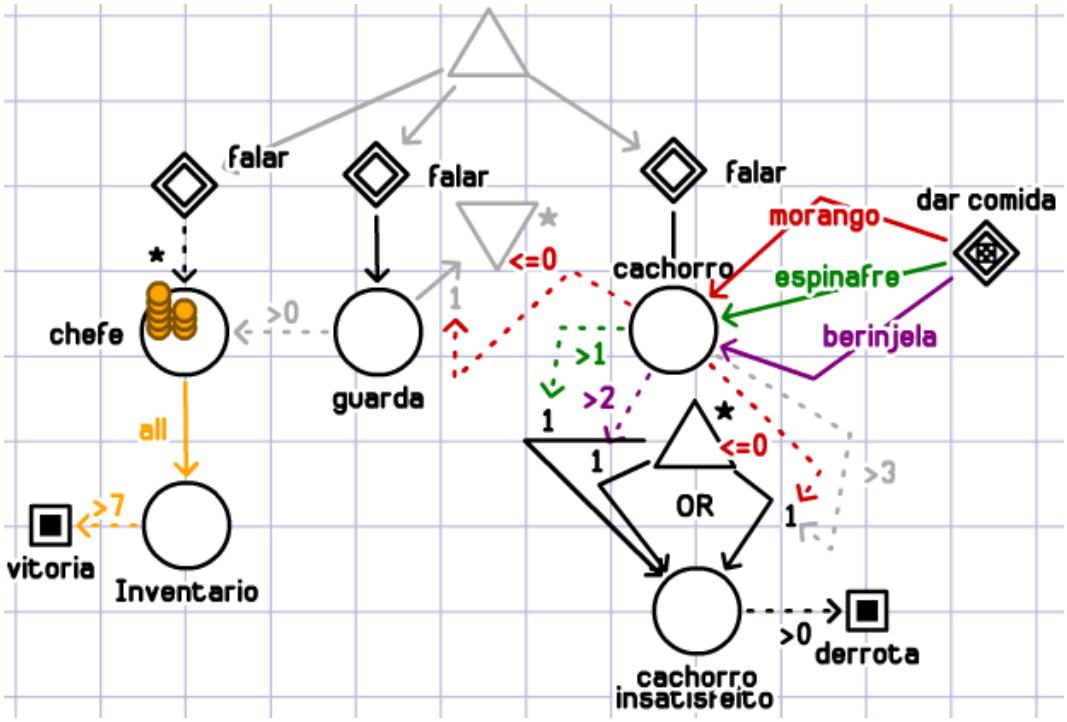


Figura 31 – Representação de interações com NPCs.

No exemplo da Figura 31 o repositório do NPC chefe guarda recursos de dinheiro que são o objetivo do jogo. Para que o jogador consiga falar com o chefe, é necessário primeiro passar pelo guarda. Do modo como está modelado, a fala do guarda permanece inalterada até que o jogador tenha agradado seu cachorro com uma comida que ele goste, neste caso o morango. Passando por esta etapa, é possível falar com o chefe e alcançar o

objetivo. As três regras de derrota são: dar mais de duas berinjelas ou dar mais que um espinafre ou falar com o cachorro mais de três vezes sem ter dado nenhum morango. Neste caso específico, é necessária a contagem de quantas vezes o jogador falou com um NPC. Outro ponto importante a ser observado foi a necessidade da definição de uma disjunção lógica. Este conceito é melhor definido na seguinte Subseção 4.1.5.

4.1.5 Conectivos lógicos

Antes de explicar conjunção e disjunção lógica, precisamos de dois conceitos básicos: proposições e conectivos lógicos. De acordo com (ROSEN, 2007), uma proposição é uma sentença declarativa (isto é, uma sentença que declara um fato) que é verdadeira ou falsa, mas não ambas. Já conectivos lógicos são operadores lógicos que são usados para formar novas proposições a partir de duas ou mais proposições existentes.

Dessa forma, em (ROSEN, 2007) as definições de **conjunção** e **disjunção lógicas** seguem da seguinte forma:

- Seja p e q proposições. A conjunção de p e q, denotada por $p \wedge q$, é a proposição "p e q". A conjunção $p \wedge q$ é verdadeira quando ambos p e q forem verdadeiros, e falsa caso contrário.

- Seja p e q proposições. A disjunção de p e q, denotada por $p \vee q$, é a proposição "p ou q". A disjunção $p \vee q$ é falsa quando ambos p e q são falsos e é verdade caso contrário.

No Machinations, ao conectar duas ou mais conexões de estado condicionais a um mesmo nó, esse nó só fica disponível caso ambas as condições sejam satisfeitas, caracterizando uma **conjunção lógica**. Portanto, conjunções são o padrão utilizado na verificação de condições conectadas a um nó no Machinations.

Existem casos em que basta uma condição satisfeita para que se queira fazer determinada ação, como foi o caso da Figura 31. Dessa forma, propõe-se um padrão para **disjunções lógicas**. Considere um nó de fonte e um nó de reservatório. Para cada condição lógica haverá uma conexão de recurso ligando a fonte ao reservatório de destino, com taxa de fluxo igual a um. Essas condições lógicas, representadas por conexões de estados, estarão ligadas a suas respectivas conexões de recurso. Por fim, saindo do reservatório de destino, haverá uma conexão de estado com condição $> \mathbf{0}$. Ou seja, se qualquer condição tiver sido verdadeira, irá ativar uma conexão de recurso que transmitirá um recurso para o reservatório. Se o reservatório tiver um recurso ou mais, é porque uma das proposições foi verdadeira, então a disjunção foi verdadeira. A Figura 32 mostra uma situação em que é necessária uma disjunção com três condições para atingir o estado de vitória. Esse estado

será alcançado se qualquer uma das condições for verdadeira. Se todas forem falsas, não é possível alcançar o estado de vitória.

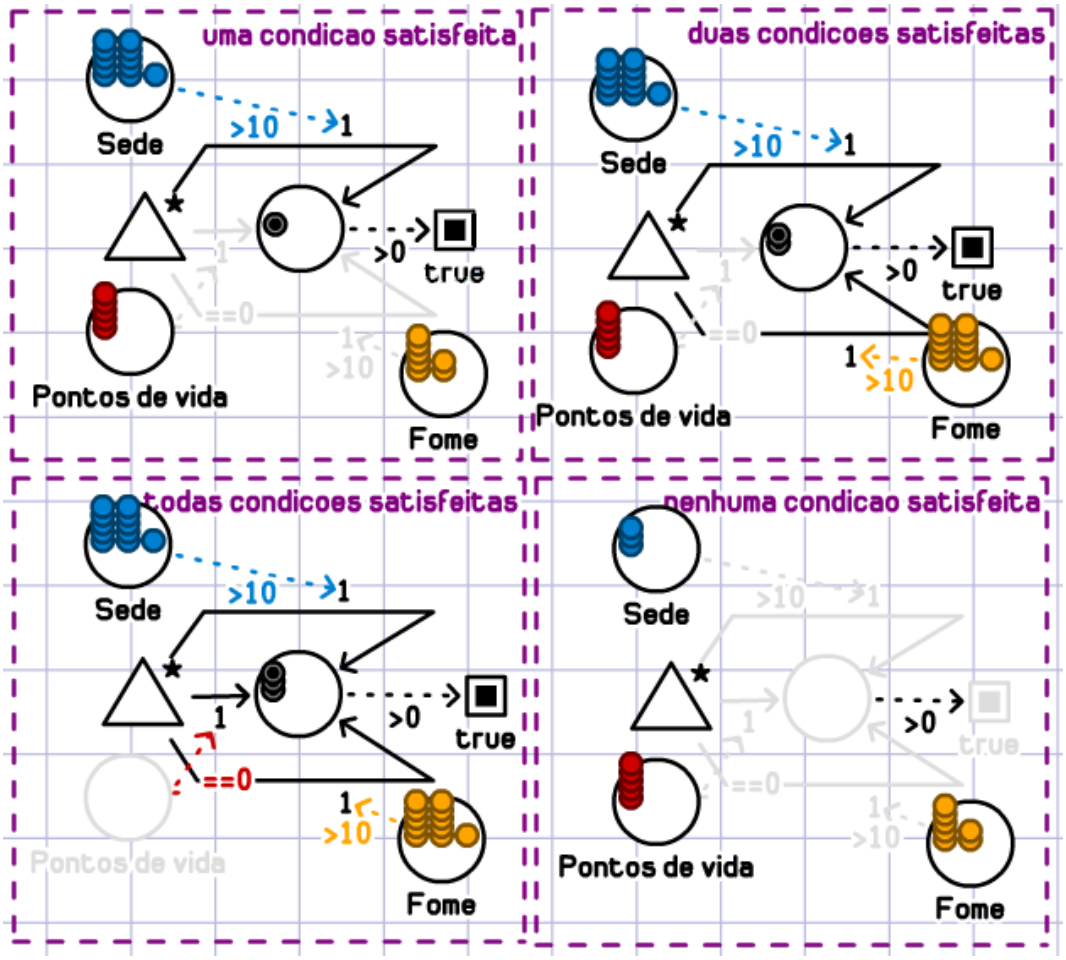


Figura 32 – Representação de uma disjunção lógica.

4.1.6 *Minigames*

Em alguns jogos de *adventure* são utilizados *minigames* para trazer maior dinamicidade e diversão. *Minigames* como o nome já diz são pequenos jogos inseridos dentro do próprio jogo. No jogo Búzios: Ecos da Liberdade existem três *minigames* que precisam ser vencidos para progredir no jogo: resolver o *puzzle* Torre de Hanói, vencer em uma queda de braço e vencer um desafio de berimbau. Um outro exemplo pode ser achado no jogo Machinarium, em que o jogador precisa vencer o *minigame 5 in a row* contra um NPC. Esses desafios geralmente não são por tentativa e erro, mas utilizam a habilidade do jogador.

Minigames podem ter uma mecânica completamente diferente de um jogo de *adventure*. Temos como exemplo os *minigames* citados acima, de tipos *puzzle*, ritmo musical e tabuleiro. Tendo em vista a variedade que *minigames* podem assumir, não

podemos especificar um passo a passo de como modelá-los, mas podemos explicar uma maneira de abstraí-los.

Uma maneira possível é utilizar o nó do tipo aleatório (*random*). Este tipo de nó produz um valor aleatório de 1 a 6. Basta especificar as probabilidades de ganhar ou perder o *minigame* através de conexões de recursos. Na Figura 33 fizemos um recorte do jogo Búzios que é modelado no próximo capítulo. Neste caso, escolhemos dar ao jogador chances iguais de perder ou ganhar o *minigame*.

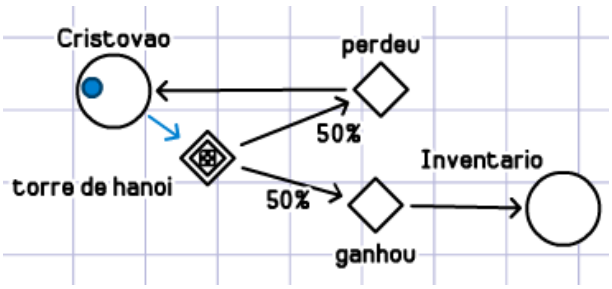


Figura 33 – Nó não-determinístico aleatório.

4.1.7 *Puzzles*

Para a solução de *puzzles* em jogos de *adventure*, geralmente são utilizados os elementos já citados nas sessões anteriores, como coleta de itens, interações com objetos, interações com NPCs, variáveis e movimentação entre cenários. Fora estes elementos, pode ser necessária a contagem de tempo também, especificada no Capítulo 3, na Seção 3.3.8.

4.1.8 *Crafting*

Em algumas situações pode ocorrer a necessidade de construir novos itens a partir de itens existentes. Esse sistema de combinação de itens é comumente chamado de sistema de *crafting* nos jogos. A Figura 34 mostra um exemplo em que é possível criar dois tipos de itens utilizando nós conversores. As conexões de entrada dos conversores são os itens necessários para o *crafting*, e as conexões de saída transmitem o item resultante para um nó de portão que nomeamos de 'Distruibuidor'. Este nó é utilizado para que não seja necessária a ligação das conexões de saída dos conversores diretamente para o inventário, deixando o diagrama mais legível. Se houver apenas um conversor, o nó 'Distribuidor' é redundante.

Essa solução é boa para combinações simples, mas modelar um sistema de *crafting* complexo que possua uma enorme variedade de itens e combinações, seria impraticável dentro do escopo do Machinations.

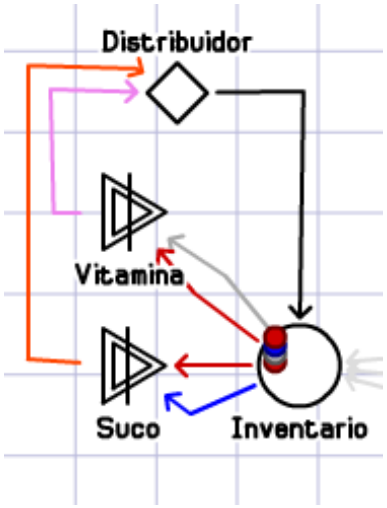


Figura 34 – Conversão de recursos.

4.2 Mecanismo de *lock and key*

Com as estruturas anteriores definidas, poderíamos modelar um simples mecanismo chamado de *lock-and-key*, exemplificado na Seção 4.3. O progresso da experiência é bloqueado (a fechadura ou *lock*) até que o jogador realize a tarefa desejada (a chave ou *key*) (ROBERTS et al., 2009). Esse tipo de mecanismo permite liberar ambientes do escopo geral do jogo aos poucos, direcionando o jogador pelos caminhos convenientes ao *game design*, fazendo com que tenha um senso de progresso e não fique perdido sem saber qual será o próximo passo. A *key* pode ser desde uma chave literal para abrir uma porta, como um obstáculo natural, um guarda que está guardando um calabouço e assim por diante. Abaixo seguem exemplos na indústria de jogos.

- A franquia Pokémon[2] na qual certas áreas são apenas desbloqueadas quando seus pokémons são ensinados através de HM's (itens chamados de *Hidden Machine*) a voar, a nadar, a utilizar luz em uma caverna escura ou a cortar árvores de bloqueio.

- Em Ori and the Blind Forest (Moon Studios, 2015) o personagem principal não tem acesso a determinadas áreas do mapa até que aprenda novas habilidades, como o pulo duplo, por exemplo.

- Em The Walking Dead Episode 1 (Telltale Games, 2012) em certo momento o jogador precisa achar as chaves da farmácia para conseguir o remédio do personagem Frank. Para isso é necessário achar o zumbi de posse das chaves e cujo corpo está preso em detritos, distrair os zumbis em volta para poder pegá-la e assim poder entrar na farmácia.

[2] Site oficial de Pokémon: http://www.pokemon.com/us/

Cenário	Objetos contidos
Sala 1	Chave azul e meia chave vermelha
Sala 2	Chave verde, gaveta com meia chave vermelha dentro e porta trancada que dá para a sala 3
Sala 3	-

Tabela 2 – Especificação da distribuição de objetos nos cenários

4.3 Exemplo básico modelado

Este exemplo foi criado especialmente para demonstrar um mecanismo de *lock-and-key* envolvendo elementos e interações que são comumente utilizados em jogos de *adventure*. Um breve resumo da distribuição dos objetos em seus ambientes segue na Tabela 2.

As seguintes regras se aplicam:

- para pegar a metade da chave vermelha na gaveta da sala 2, é necessário antes abrir a gaveta;
- as metades de chaves vermelhas se combinam para formar uma única e completa chave vermelha;
- o jogador tem como posição inicial a sala 1;
- o jogador só pode interagir ou coletar objetos estando na sala em que eles se encontram;
- para abrir a porta que conecta a sala 2 até a sala 3 e mudar de sala, é necessário que o jogador possua a chave azul, a chave vermelha, esteja na sala 2 e tentou abrir a porta;
- o jogo termina quando o jogador alcança a sala 3, sendo este o objetivo principal.

O diagrama completo modelado no Machinations pode ser observado na Figura 35.

As chaves são itens coletáveis de cenário, portanto deverão ser modeladas como recursos que devem ser transmitidos para o inventário do jogador ao serem coletados. Como apenas as chaves vermelha e azul abrem a porta para a sala 3, mas a chave verde não, elas foram modeladas com cores diferentes para as condições serem devidamente modeladas. Se a chave vermelha e azul possuem a mesma propriedade de abrir a porta, elas poderiam ser modeladas com a mesma cor, mas para fins de ilustração foram especificadas com suas respectivas cores. Para a criação da chave vermelha a partir das metades de chave vermelha, foi utilizado um conversor.

Os nós de reservatórios não tem a opção de definir recursos de cores distintas em sua composição inicial, então surgiu a necessidade de criar o padrão **Fábrica de itens**.

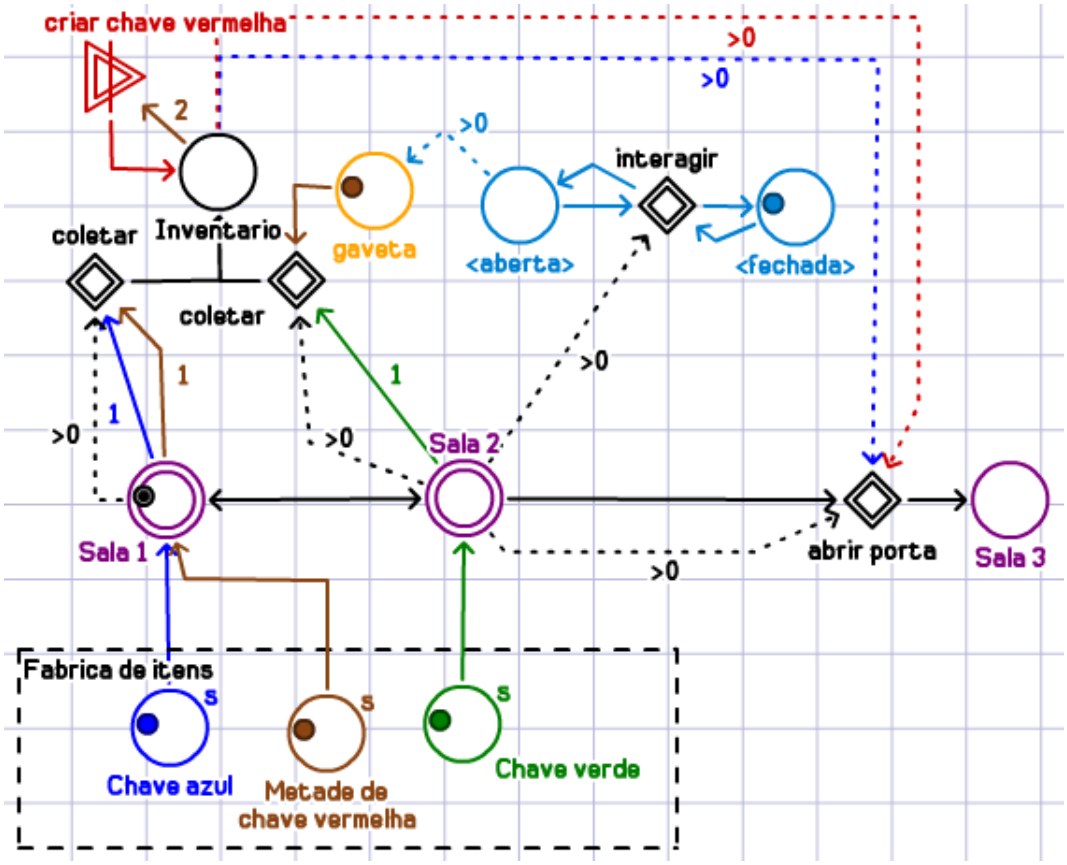


Figura 35 – Exemplo de diagrama utilizando o mecanismo de *lock-and-key*.

A GroupBox é opcional, mas é recomendada para melhor visualização. Dentro dela, são colocados nós com modo de ativação inicial, que irão transmitir os itens para seus devidos cenários no início do jogo.

Já a gaveta e a porta são objetos interagíveis modelados de formas diferentes. Como neste caso o objetivo final do jogo é chegar na sala 3, não precisamos dos estados de porta aberta ou fechada. Basta apenas a ação de abrir porta, representada por um portão. Quando todas as suas condições são atingidas, este nó é ativado, tornando-se interagível para o jogador. No caso da gaveta, ela foi modelada como um reservatório, pois ela guarda o recurso de metade de chave vermelha. Porém esse recurso só pode ser coletado caso a gaveta tenha sido aberta. Para isso, cria-se um nó de portão que representará a interação de abrir ou fechar a gaveta. Dois reservatórios foram criados para representar os estados de gaveta aberta e gaveta fechada. Um recurso dentro deste reservatório pode ser comparado ao valor lógico **true** atribuído a uma variável de programação. Como são duas variáveis antônimas, ou seja, uma gaveta não pode estar aberta e fechada ao mesmo tempo, basta um recurso para transitar entre os dois reservatórios. Desta forma, podemos representar quaisquer variáveis binárias em um jogo.

5 Avaliação

Neste capítulo a avaliação deste trabalho será feita de duas formas. A primeira será mostrar ser possível modelar um jogo de adventure publicado e validado utilizando os passos sugeridos no Capítulo 4.

Já na segunda parte faremos uma comparação qualitativa entre a utilização de redes de Petri para modelar uma IF, o Cloak of Darkness, e a abordagem feita no Machinations.

5.1 Estudo de caso 1: fase do convés do jogo Búzios

Existem duas características diferentes do exemplo da Seção 4.3 que valem ser mencionadas.

- Toda a fase se passa em um único cenário, o convés do navio. Não existe a necessidade de modelar onde o jogador se encontra fisicamente.
- Existem NPCs com os quais o jogador deve dialogar (interagir) para progredir na fase.

Tendo em vista essas duas características, o diagrama focará na interação com os personagens. Mas antes de explicar sua construção, vamos contextualizar o exemplo de estudo.

Nessa fase, o jogo se inicia com Francisco (o jogador principal) indo para Salvador em um navio português. Ele precisa pegar sua mala que está no porão, mas a sua passagem está bloqueada pelo NPC **Roderico**. Para que ele consiga passar, primeiramente ele tem que saber que a mala dele está no porão, informação que é obtida falando com o NPC **Pierre**. Após falar com Pierre, deve-se falar com o NPC **Sebastião**, que lhe ajudará, mas para isso Francisco precisa lhe fazer um favor. Este favor é pegar suas **porcelanas** de volta na mão do NPC **Cristovão**. Para pegar as porcelanas com Cristovão, é necessário ganhar o *minigame* de **Torre de Hanói**. Vencendo o desafio, ao falar novamente com Sebastião, este vai lhe dar um **laxante** para pôr na bebida de Roderico, em troca pelas porcelanas. Ao falar com Roderico, deve-se escolher a opção de diálogo que é uma história contada por Francisco, a qual assusta Roderico e o distrai. Enquanto está distraído, pode-se combinar o laxante com a bebida de Roderico. Logo após, Roderico a bebe, fica com dor de barriga e libera a entrada para o porão.

Como especificado na Subseção 4.1.4, cada NPC terá um reservatório para si onde armazenará seus recursos de diálogo e possíveis itens que estejam sob sua posse. Ações

do jogador foram modeladas como descrito na Subseção 4.1.3 com nós de portão, sendo estas ações as de falar, jogar o Torre de Hanói, pôr o laxante na bebida de Roderico e interagir com a porta do porão. Exceto os casos dos reservatórios de NPCs, os portões foram basicamente conectados a reservatórios que indicam estados, como ganhou ou não o *minigame*, se Roderico está distraído, normal ou com problemas intestinais, ou se a porta está aberta ou fechada. Vale observar mais duas estruturas importantes utilizadas: um nó trocador e um nó de atraso. O nó trocador foi utilizado para a troca dos itens porcelanas e laxante com Sebastião. Já o nó de atraso é comumente utilizado para mecânicas que envolvem a contagem de tempo, e neste exemplo teve como propósito contar o tempo que Roderico ficaria no estado de distraído antes de voltar para o seu estado padrão.

Para o funcionamento do *minigame* Torre de Hanói utilizou-se apenas um nó de portão com propriedade aleatória, tendo 50% de chance de ganhar e 50% de perder. Não houve necessidade de modelar qual opção de diálogo foi escolhida entre as opções da conversa com Roderico, a qual levaria o NPC a se distrair e ficar vulnerável. Desta forma foi aplicada uma condição de na quarta vez que o jogador interage com Roderico, ele entra em estado de distraído durante 3 segundos. Veja o diagrama completo na Figura 37.

Figura 36 – Cena do convés com o jogador Francisco e os NPCs Roderico e Cristovão.

A Figura 36 mostra uma captura de tela da fase do convés, estando presentes nela os personagens Roderico o qual está guardando a porta do porão, Cristovão e o jogador principal Franciso ao centro.

5.2 Estudo de caso 2: Cloak of Darkness

Ao contrário do jogo Búzios: Ecos da Liberdade que se enquadra em um jogo do tipo *point-and-click*, ou seja as interações são feitas por meio do mouse, o Cloak of Darkness é um IF tendo como entrada comandos textuais.

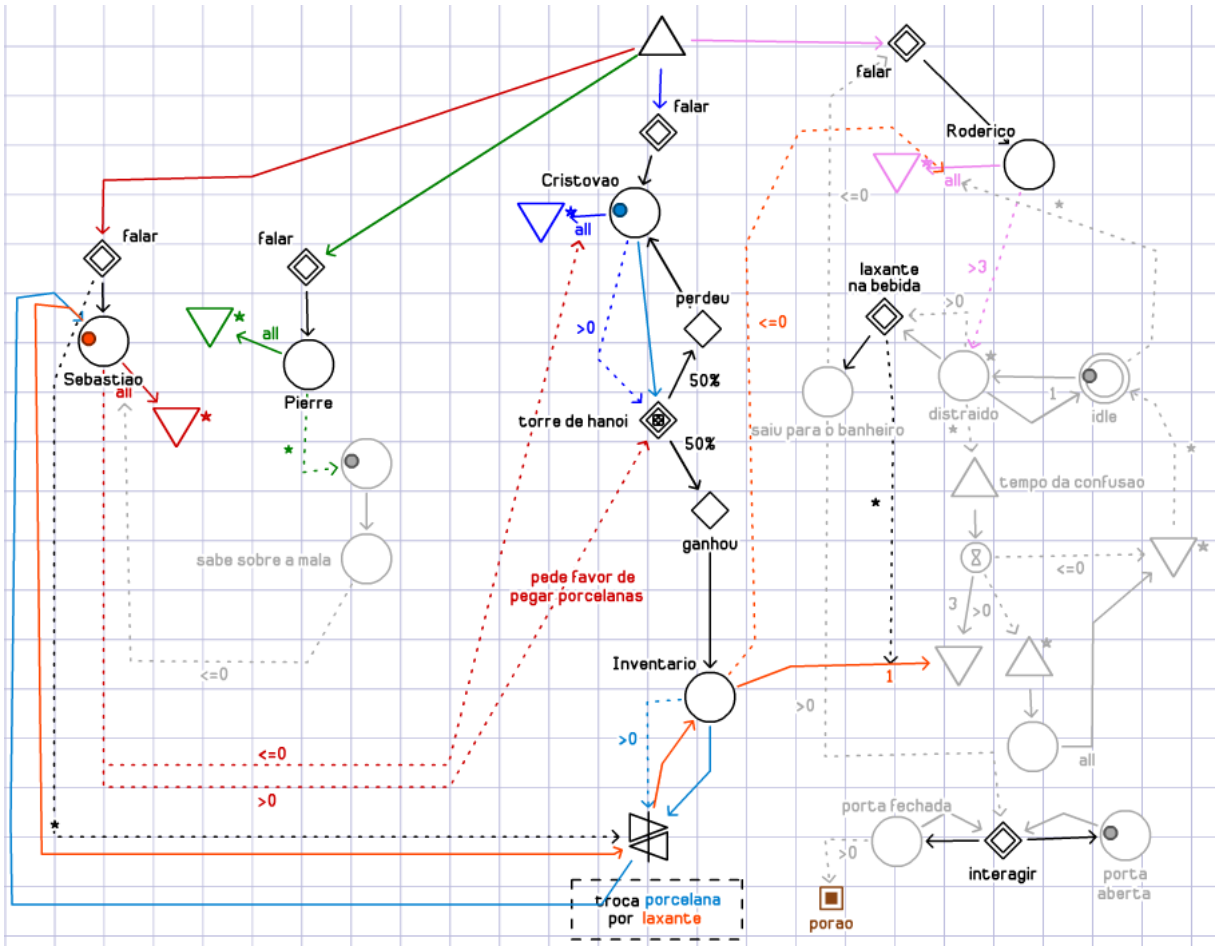


Figura 37 – Diagrama completo da fase do convés do jogo Búzios.

As especificações são descritas a seguir:

- Os cenários possíveis são: *Foyer* (hall de entrada), *Bar* e *Cloakroom* (vestiário).
- O jogador inicia no *Foyer*. Este ambiente possui portas que dão para cenários ao sul e ao oeste. Há uma saída para o norte, porém interditada. Não existe ninguém por perto.
- O *Bar* localiza-se ao sul de *Foyer* e está inicialmente escuro. Qualquer coisa que se faça estando nessa ambiente enquanto escuro, sem ser voltar para o norte, resulta em um alerta sobre perturbar coisas no escuro.
- Na parede do *Cloakroom*, que fica ao oeste de *Foyer*, existe um pequeno gancho de bronze fixado.
- Ao examinar o inventário, o jogador percebe que está vestindo um manto preto de veludo. Examinando este manto, percebe também que ele absorve luz. Dentro do *Cloakroom* o jogador pode soltar o manto no chão, ou melhor, colocá-lo no gancho.

- Voltando ao Bar sem o manto, revela-se que está agora iluminado. Uma mensagem é mostrada no chão. A mensagem é lida como "Você ganhou"ou "Você perdeu", dependendo de quantas vezes o Bar foi perturbado pelo jogador enquanto a sala estava escura.

- O ato de ler a mensagem termina o jogo.

Na Figura 38 foram realizados os seguintes passos para chegar no estado de vitória:

```
> go west - vá para o oeste
> examine brass hook - analise o gancho de bronze
> inventory - inventário
> examine black velvet cloak - analise manto preto de veludo
> put cloak on the hook - coloque o manto no gancho
> go east - vá para o leste
> go south - vá para o sul
> read message - leia a mensagem
```

5.2.1 Modelagem utilizando Machinations

As informações de localização dos cenários são descritas na especificação do jogo e foram modeladas no trabalho de (SOUZA, 2008) que será utilizado na Seção 5.3 a fim de comparação. Portanto, além de todas as ações serem modeladas como portões, as ações de movimentação direcionais também serão modeladas da mesma forma. Os cenários foram modelados com reservatórios de cor azul e o manto como um recurso verde iniciando no inventário. Este manto só pode ser colocado no gancho quando o jogador estiver no Cloakroom, assim como ser guardado de volta. Qualquer ação diferente de sair da sala Bar foi modelada como um portão que ao interagir, aciona um nó de fonte e gera recursos vermelhos de perturbação. Caso essa quantidade seja for maior do que a especificada, o jogador perde o jogo. Essa modelagem se encontra na Figura 39.

5.2.2 Modelagem utilizando rede de Petri

Agora que o Cloak of Darkness foi devidamente modelado no Machinations, podemos fazer um estudo comparativo com uma modelagem proposta por (SOUZA, 2008) utilizando redes de Petri. A rede foi dividida em três partes sendo: a primeira representada na Figura 40 mostrando a movimentação entre as salas; a segunda representada na Figura 41 e a terceira na Figura 42, mostrando as ações disponíveis nas salas Cloakroom e Bar, respectivamente.

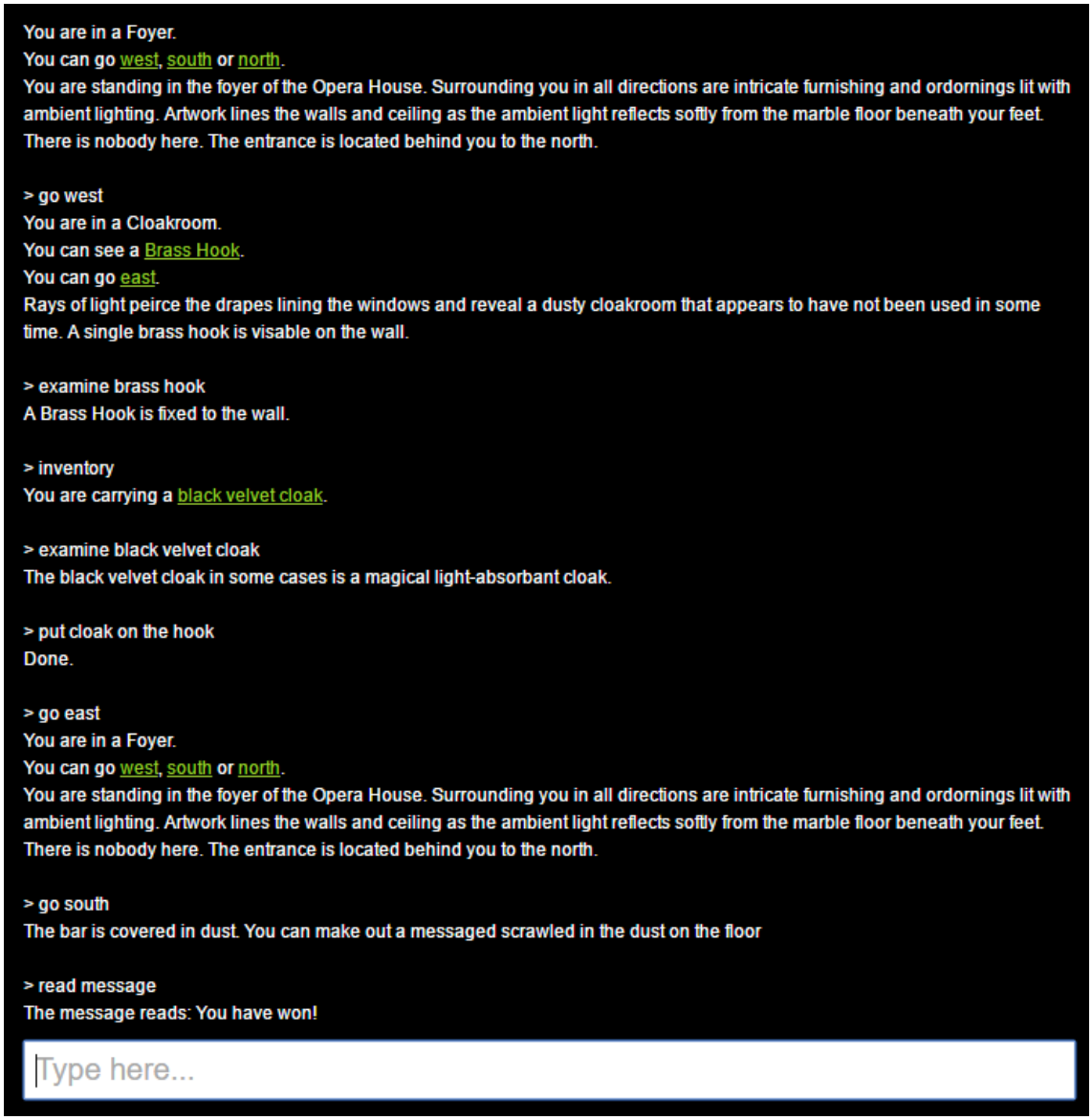


Figura 38 – Captura de tela do passo a passo do Cloak of Darkness.

De acordo com (SOUZA, 2008), uma rede de Petri colorida é composta de lugares (*places*), transições (*transitions*) e fichas (*tokens*). Cada lugar contém fichas de um determinado tipo (*colour set*). Transições, quando disparadas, removem fichas contidas em certos lugares e inserem novas fichas em determinados lugares.

Os círculos verdes representam o mesmo lugar PlayerRoom, mas suas transições só serão habilitadas caso sua ficha específica esteja no lugar PlayerRoom.

5.3 Discussão

O primeiro ponto que podemos observar em termos numéricos é a quantidade de elementos nos diagramas. Contabilizando os elementos utilizados no Machinations temos

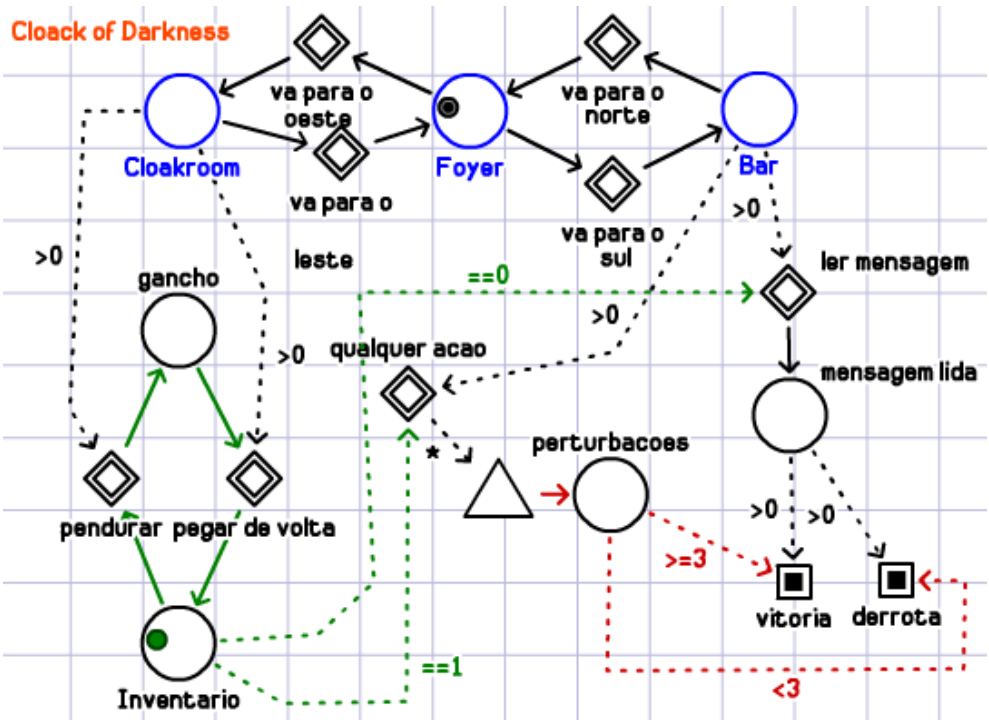


Figura 39 – Cloak of Darkness no Machinations.

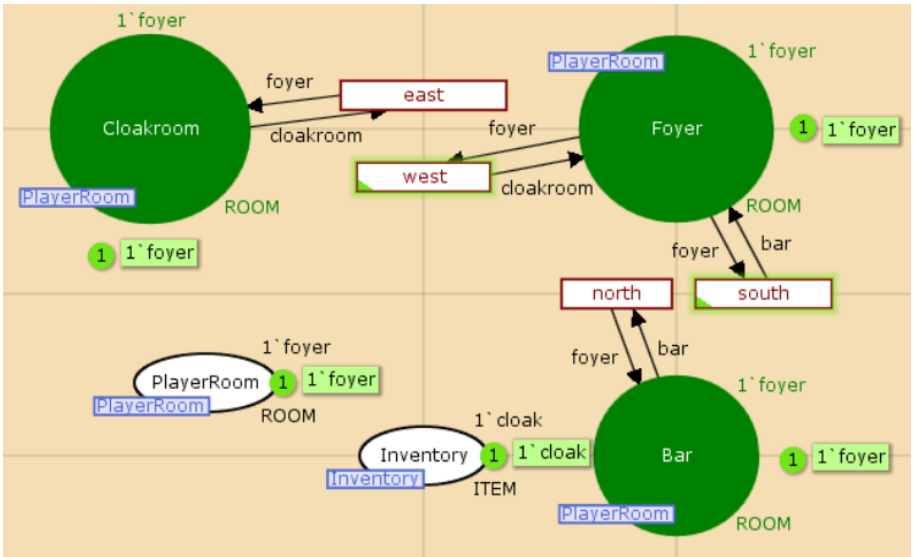


Figura 40 – Movimentação entre salas na modelagem de Cloak of Darkness em uma rede de Petri por (SOUZA, 2008).

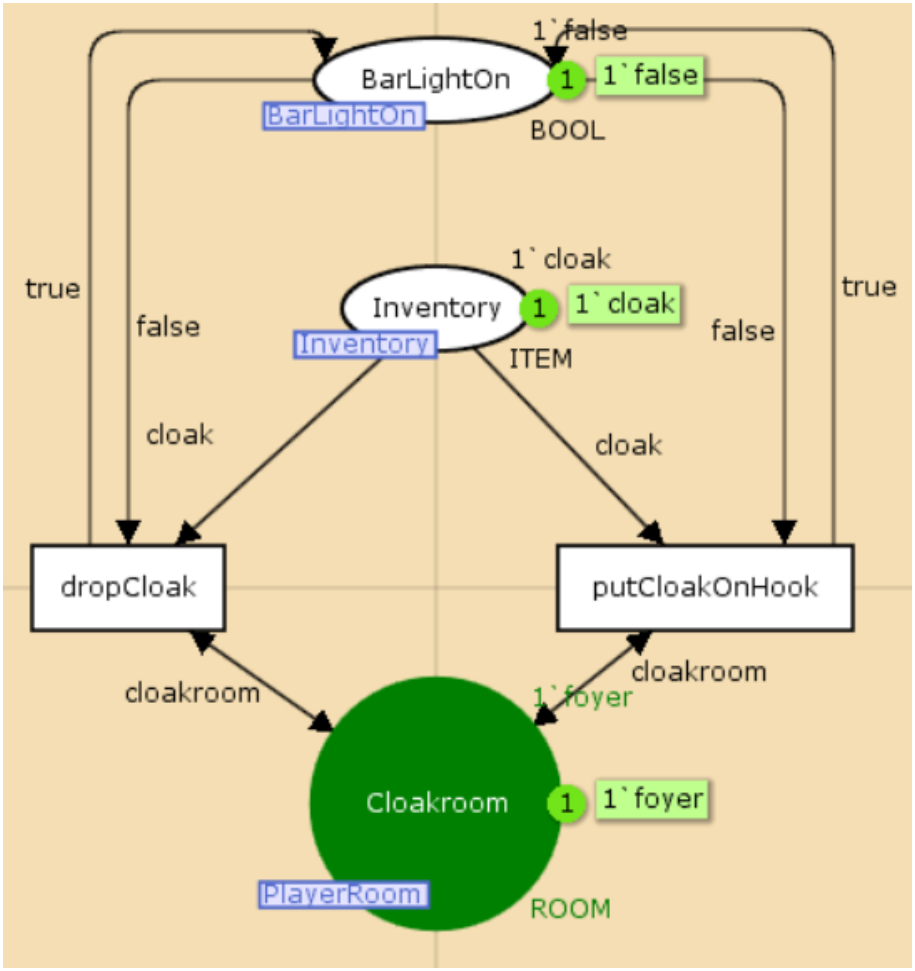


Figura 41 – Possíveis ações no ambiente Cloakroom modeladas por (SOUZA, 2008).

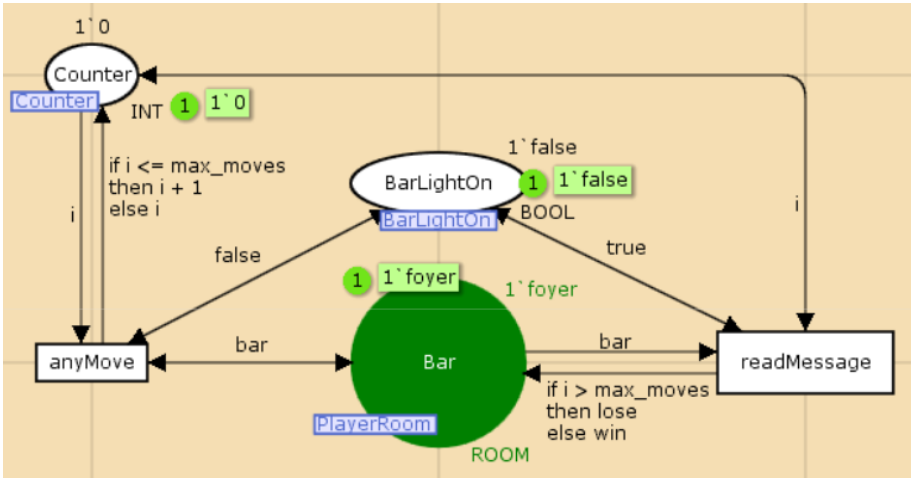


Figura 42 – Possíveis ações no ambiente Bar modeladas por (SOUZA, 2008).

como resultado um total de 18 nós e 25 conexões. Já na rede de Petri, contabilizamos 15 nós se formos considerar transições e lugares como nós e desconsiderar os lugares que foram repetidos na divisão da rede para melhor visualização. Considerando arcos bidirecionais temos um total de 29 arcos (equivalentes a conexões). Como a diferença entre a quantidade de elementos não foi grande, para este exemplo específico não foi um fator decisivo.

Outro ponto relevante é a diferença entre *tokens* e recursos. Eles desempenham um papel semelhante, mas as cores dos *tokens* na verdade são tipos especificados na modelagem, como a ficha "foyer" representando a sala Foyer. Já no Machinations, os recursos possuem apenas cores diversas, mas não um tipo. Então se formos considerar uma situação macro em que seja a necessária a representação de variados tipos de itens, por exemplo, a diferenciação desses itens fica muito mais clara com redes de Petri. Em contrapartida, no Machinations é possível utilizar cores em todos elementos, incluindo conexões, textos e nós, o que torna a visualização muito mais ilustrativa.

Como as Redes de Petri provém de uma estrutura matematicamente bem fundada, elas podem ser verificadas, validadas e simuladas com uma série de métodos e ferramentas de análise (ARAÚJO; ROQUE, 2009). No trabalho de (SOUZA, 2008), foi utilizada a ferramenta CPN Tools (ferramenta de modelagem, simulação e verificação de redes de Petri coloridas) para gerar o espaço de estados do Cloak of Darkness, bem como realizar uma simulação controlada e aplicar métricas definidas de avaliação. A ideia básica por trás de espaços de estados é calcular todos os estados alcançáveis e mudanças de estado do sistema, e representá-los como um grafo direcionado (KRISTENSEN, 2000). O *framework* Machinations não possui uma implementação que gere o espaço de estados de um diagrama. Porém o trabalho especificado na Seção 3.4 aborda uma análise formal através de Micro-Machinations. Visto que o Micro-Machinations é um subconjunto estendido do Machinations, não podemos aplicar seus métodos de avaliação formal para o nosso estudo de caso modelado.

Para modelar sistemas mais complexos no Machinations, a variedade de nós e suas funcionalidades facilita na construção dos diagramas. Mas devido a essa variedade, pessoas que nunca tenham tido contato com ambas as abordagens possivelmente terão maior facilidade de entendimento com redes de Petri. Como a quantidade de elementos diferentes no Machinations é maior que na rede de Petri, é provável que demande um maior esforço de aprendizado de sua estrutura.

6 Conclusão

Neste trabalho realizamos a proposta de mapeamento de jogos de *adventure* na linguagem visual Machinations, e realizamos a modelagem de três jogos simples. Tivemos como contribuições:

- confirmação que é factível a modelagem de jogos de *adventure* no Machinations, mesmo que este não seja o estilo de jogo focado em suas modelagens, já que não existe uma economia de jogo complexa;
- criação de conteúdo em português sobre o Machinations, dada a escassez de materiais nesta língua;
- comparação qualitativa da modelagem de um *adventure* no Machinations e em redes de Petri.

Percebemos ao longo deste trabalho que existe certa deficiência na área de pesquisa de ferramentas e linguagens para modelagem na área de game design. Esta é uma área grande e complexa, que envolve engenharia de software, métodos de avaliações formais matemáticas e game design. Jogos são complexos, possuem variados gêneros e desenvolver uma linguagem que atenda tais requisitos se torna um grande desafio. Uma possibilidade é focar a pesquisa e desenvolvimento para um gênero específico, mas ainda sim não torna esta tarefa trivial. Além dos fatores citados, existem fatores de resistência pela própria indústria de jogos. Aprender uma ferramenta demanda esforço dos desenvolvedores, e até então nada garante que ela atenda todas as especificações necessárias.

6.1 Práticas para melhoria de legibilidade

Durante o desenvolvimento deste trabalho, percebeu-se que é exigido um certo gasto de energia para modelar os diagramas de forma organizada e visualmente limpa. Quanto maior a quantidade de interseções entre conexões, mais o diagrama fica confuso. Algumas práticas podem auxiliar na organização, como:

- Analisar se um nó que se conecta com 'n' nós deve ser dividido em 'n' nós. Podemos exemplificar a partir da Figura 37. Os drenos utilizados para destruir recursos de fala dos NPCs eram inicialmente um nó único conectando a todos. Mas juntando com todos os outros elementos do diagrama, verificou-se que a legibilidade do diagrama foi prejudicada, contradizendo a proposta de um modelo de fácil visualização e

entendimento. A criação de um dreno para cada NPC melhorou significantemente a organização do diagrama.

- Utilização de cores para diferenciar elementos do diagrama, mesmo que o recurso de *color coding* não seja utilizado.
- Utilização de elementos meramente ilustrativos para melhorar a compreensão do contexto do diagrama, como o TextLabel e o GroupBox.

6.2 Trabalhos futuros

Com base na confirmação da viabilidade da modelagem de jogos de *adventure* no Machinations, consideramos como um possível trabalho futuro a incorporação da MM Lib em um protótipo de um jogo de *adventure*, podendo assim verificar a sua aplicabilidade e fazer uma análise formal detalhada.

Seria interessante a participação da comunidade para o desenvolvimento de algumas melhorias na ferramenta citadas a seguir.

- Escolha de cores pelo sistema RGB, pois além de poupar a necessidade de digitar o nome da cor, teria-se uma variedade de 256^3 cores.
- Reservatórios terem a possibilidade de inicialmente conterem recursos de mais de uma cor.
- O elemento GroupBox poderia ter uma nova funcionalidade ao invés de ser meramente ilustrativo. Este elemento agruparia itens dentro de si, mas tendo a possibilidade de encapsular seu conteúdo ou expandir para mostrá-lo. Isso facilitaria tanto a legibilidade do diagrama, como criaria mais espaço. Uma sugestão de como seria pode ser observada na Figura 43.

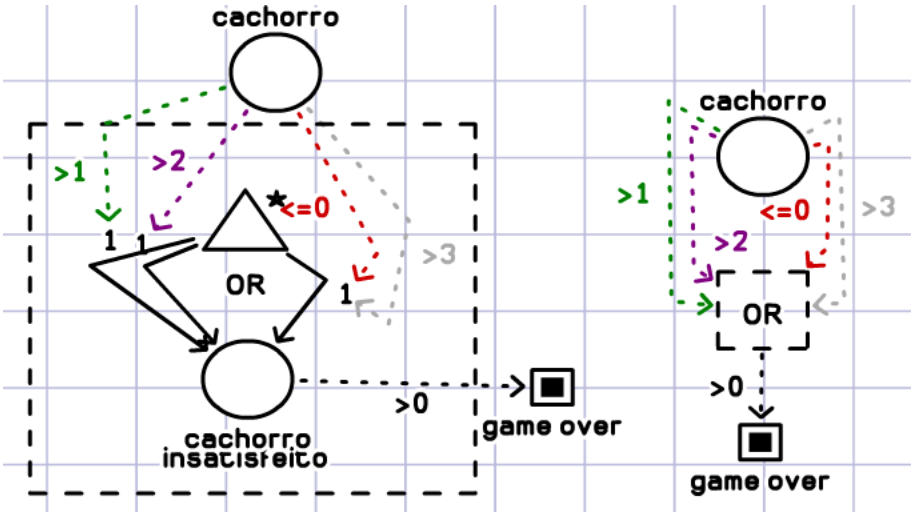


Figura 43 – GroupBox modelado de forma agrupada

Bibliografia

ADAMS, E. *Fundamentals of Game Design*. 2nd. ed. Thousand Oaks, CA, USA: New Riders Publishing, 2009. ISBN 0321643372, 9780321643377. Citado 3 vezes nas páginas 19, 20 e 23.

ADAMS, E.; DORMANS, J. *Game Mechanics: Advanced Game Design*. 1st. ed. Thousand Oaks, CA, USA: New Riders Publishing, 2012. ISBN 0321820274, 9780321820273. Citado 5 vezes nas páginas 19, 20, 27, 33 e 34.

Amanita Design. *Machinarium*. Amanita Design, 2009. Disponível em: <http://machinarium.net/>. Citado na página 23.

ARAÚJO, M.; ROQUE, L. Modeling Games with Petri Nets. In: ATKINS, B.; KENNEDY, H.; KRZYWINSKA, T. (Ed.). *Breaking New Ground: Innovation in Games, Play, Practice and Theory: Proceedings of the 2009 Digital Games Research Association Conference*. London: Brunel University, 2009. Disponível em: <http://www.digra.org:8080/Plone/dl/display_html?chid=09287.37256.pdf>. Citado 2 vezes nas páginas 26 e 60.

BALAS, D. et al. Hierarchical petri nets for story plots featuring virtual humans. In: *Proceedings of the Fourth AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*. AAAI Press, 2008. (AIIDE'08), p. 2–9. Disponível em: <http://dl.acm.org/citation.cfm?id=3022539.3022541>. Citado na página 26.

BROM, C.; SISLER, V.; HOLAN, T. Story manager in 'europe 2045' uses petri nets. In: *Proceedings of the 4th International Conference on Virtual Storytelling: Using Virtual Reality Technologies for Storytelling*. Berlin, Heidelberg: Springer-Verlag, 2007. (ICVS'07), p. 38–50. ISBN 3-540-77037-2, 978-3-540-77037-4. Disponível em: <http://dl.acm.org/citation.cfm?id=1777851.1777856>. Citado na página 26.

CAMURUGY, L. et al. Relatos de caminho trilhado para o desenvolvimento de um jogo de adventure no flash. In: *Brazilian Symposium on Computer Games and Digital Entertainment (SBGAMES) - Computing Track*. Florianópolis, Brazil: SBC, 2010. p. 328–332. Citado na página 23.

Comunidades Virtuais. *Búzios: Ecos da Liberdade*. 2011. Disponível em: <http://www.comunidadesvirtuais.pro.br/buzios/>. Citado na página 24.

COOK, D. *The Chemistry of Game Design*. 2007. Article in the Gamasutra webzine. Disponível em: <http://www.gamasutra.com/view/feature/129948/the_chemistry_of_game_design.php?page=7>. Citado na página 26.

Cyan. *Myst*. Brøderbund, 1993. Disponível em: <http://cyan.com/games/myst/>. Citado na página 23.

Dontnod Entertainment. *Life is Strange*. Square Enix,Feral Interactive, 2015. Disponível em: <http://www.lifeisstrange.com>. Citado na página 23.

DORMANS, J. Machinations: Elemental Feedback Patterns for Game Design. In: SAUR, J.; LOPER, M. (Ed.). *GAME-ON-NA 2009: 5th International North American Conference on Intelligent Games and Simulation*. [s.n.], 2009. p. 33–40. Disponível em: <http://www.jorisdormans.nl/article.php?ref=machinations>. Citado 3 vezes nas páginas 20, 40 e 43.

Infocom. *Zork*. [S.l.]: Personal Software, Infocom, Activision, 1980. Citado na página 23.

JU, E.; WAGNER, C. Personal computer adventure games: Their structure, principles, and applicability for training. *SIGMIS Database*, ACM, New York, NY, USA, v. 28, n. 2, p. 78–92, abr. 1997. ISSN 0095-0033. Disponível em: <http://doi.acm.org/10.1145/264701.264707>. Citado na página 25.

KASURINEN, J. Games as software: Similarities and differences between the implementation projects. In: *Proceedings of the 17th International Conference on Computer Systems and Technologies 2016*. New York, NY, USA: ACM, 2016. (CompSysTech '16), p. 33–40. ISBN 978-1-4503-4182-0. Disponível em: <http://doi.acm.org/10.1145/2983468.2983501>. Citado na página 19.

KASURINEN, J.; STRANDÉN, J.-P.; SMOLANDER, K. What do game developers expect from development and design tools? In: *Proceedings of the 17th International Conference on Evaluation and Assessment in Software Engineering*. New York, NY, USA: ACM, 2013. (EASE '13), p. 36–41. ISBN 978-1-4503-1848-8. Disponível em: <http://doi.acm.org/10.1145/2460999.2461004>. Citado na página 19.

KLINT, P.; ROZEN, R. V. Micro-Machinations: A DSL For Game Economies. In: ERWIG, M.; PAIGE, R. F.; WYK, E. van (Ed.). *Proceedings of the International Conference on Software Language Engineering (SLE, 2013)*. Unknown, United States: Springer, 2013. (Lecture Notes in Computer Science, v. 8225), p. 36 – 55. Disponível em: <https://hal.inria.fr/hal-00923383>. Citado 3 vezes nas páginas 40, 41 e 42.

KRISTENSEN, L. M. *State Space Methods for Coloured Petri Nets*. Tese (Doutorado) — University of Aarhus, 1 2000. Citado na página 60.

Lucasfilm Games. *The Secret of Monkey Island*. [S.l.]: Lucasfilm Games, 1990. Citado na página 23.

Moon Studios. *Ori and the Blind Forest*. Microsoft Studios, 2015. Disponível em: <http://www.oriblindforest.com/>. Citado na página 50.

ROBERTS, D. L. et al. Using influence and persuasion to shape player experiences. In: *Proceedings of the 2009 ACM SIGGRAPH Symposium on Video Games*. New York, NY, USA: ACM, 2009. (Sandbox '09), p. 23–30. ISBN 978-1-60558-514-7. Disponível em: <http://doi.acm.org/10.1145/1581073.1581077>. Citado na página 50.

ROSEN, K. H. *Discrete mathematics and its applications*. [S.l.: s.n.], 2007. v. 10. 12 p. Citado na página 47.

ROZEN, R. van; DORMANS, J. Adapting game mechanics with micro-machinations. In: SOCIETY FOR THE ADVANCEMENT OF THE SCIENCE OF DIGITAL GAMES. *Proceedings of the 9th International Conference on the Foundations of Digital Games*. [S.l.], 2014. Citado na página 42.

Sierra On-Line. *King's Quest*. [S.l.]: IBM, Sierra On-Line, Parker Brothers (SMS), 1984. Citado na página 23.

SOUZA, R. R. G. *Modelagem e verificação de jogos estilo adventure através de redes de Petri*. [S.l.], 2008. 11 p. Citado 7 vezes nas páginas 21, 26, 56, 57, 58, 59 e 60.

TAYLOR, M. J.; GRESTY, D.; BASKETT, M. Computer game-flow design. *Comput. Entertain.*, ACM, New York, NY, USA, v. 4, n. 1, jan. 2006. ISSN 1544-3574. Disponível em: <http://doi.acm.org/10.1145/1111293.1111300>. Citado na página 25.

Telltale Games. *The Walking Dead*. Telltale Games, 2012. Disponível em: <https://telltale.com/series/the-walking-dead/>. Citado 2 vezes nas páginas 23 e 50.

Telltale Games. *Tales From The Borderlands*. Telltale Games, 2014. Disponível em: <https://telltale.com/series/tales-from-the-borderlands/>. Citado na página 23.

The Fullbright Company. *Gone Home*. The Fullbright Company, 2013. Disponível em: <https://gonehome.game/>. Citado na página 23.

VERBRUGGE, C.; ZHANG, P. Analyzing computer game narratives. In: *Entertainment Computing – ICEC 2010, 9th International Conference*. [S.l.]: Springer-Verlag, 2010. (LNCS, 6243), p. 224–231. Citado 2 vezes nas páginas 23 e 26.

William Crowther and Don Woods. *Colossal Cave Adventure*. 1976. Citado na página 23.