Modelagem de jogos de *adventure* através de Machinations

First author* Second author Third author

Research institute X

Figura 1: Machinarium, jogo de *adventure* desenvolvido pela Amanita Design, 2009.

RESUMO

Projetar jogos de *adventure*, onde jogadores podem executar ações em diferentes sequências para progredir no jogo, é um desafio para game designers. 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. A partir do uso destes padrões, foram modeladas partes de jogos de *adventure* com Machinations, levando-nos à conclusão de que é factível esta abordagem mesmo que tal gênero não seja o foco do *framework*. Adicionalmente concluímos que existe uma carência de boas ferramentas para modelagens de jogos e de pesquisa acadêmica na área.

Keywords: jogos de *adventure*, modelagem formal, Machinations.

1 INTRODUÇÃO

Inaugurado com o jogo Colossal Game Adventure (1976), o gênero *adventure* é caracterizado por exploração, coleta e manipulação de itens, e resolução de *puzzles*. Inicialmente os jogos de *adventure* eram compostos apenas de texto, e eram conhecidos também como ficção interativa (*interactive fiction* ou IF); mais tarde surgiram os jogos de *adventure* gráficos jogados com o auxílio do *mouse*, motivo pelo qual esses jogos se tornaram conhecidos como *point & click* (apontar e clicar).

Jogos de *adventure* fizeram muito sucesso até os anos 2000; uma demonstração disso é o *adventure* Myst (Cyan Inc., 1993), o jogo mais vendido para PC até 2002.Embora sua popularidade tenha diminuído, o gênero *adventure* continua se transformando e influenciando outros gêneros de jogos. Exemplos de sucessos comerciais recentes incluem Machinarium (Amanita Design, 2009), ilustrado na Figura 1, The Walking Dead (Telltale Games, 2012) e Life is Strange (Dontnod Entertainment, 2015).

Criar bons jogos de *adventure*, sem problemas graves de design, é um desafio. Para citar um exemplo, alguns jogos de *adventure* mais antigos possuíam *becos sem saída*: situações nas quais o jogador não consegue progredir no jogo por causa de alguma ação que ele realizou anteriormente. No jogo King's Quest (Sierra Entertainment, 1984), por exemplo, o jogador podia, no início do jogo, descartar um item que é necessário para resolver um desafio em um momento posterior e, assim, anular qualquer chance de continuar progredindo no jogo.

Hoje considera-se que é uma boa prática evitar becos sem saída ao projetar jogos. Essa não é uma tarefa trivial, no entanto. Ao mesmo tempo que o *game designer* deve dar liberdade ao jogador, possibilitando a ele resolver desafios em diferentes sequências, ele deve garantir que o jogador sempre consegue progredir, independentemente da ordem na qual resolveu os desafios.

A prototipagem é um recurso essencial para o *game designer*, pois permite que ele simule e avalie ideias para um jogo de forma rápida, ainda no estágio de pré-produção. Existem poucas ferramentas, no entanto, para apoiar o *game design* de jogos de *adventure*.

Machinations é um *framework* conceitual, uma linguagem visual e uma ferramenta focada em descrever e simular economias internas de jogos. Ela é baseada em formalismos como redes de Petri e autômatos finitos. Machinations tem sido usado em diversos livros e artigos sobre *game design* [2, 15].

Ainda que jogos de *adventure* não sejam caracterizados pela presença de economias internas, usar Machinations para modelar esses jogos pode fazer sentido pelo fato de ser um *framework* de uso difundido. Neste trabalho investigamos a viabilidade do uso de Machinations para modelar e simular jogos de *adventure*. Para atingir esse objetivo geral, buscamos desenvolver padrões para modelagem de elementos de jogos de *adventure* usando Machinations e avaliar a aplicabilidade dos padrões a jogos de *adventure* existentes; neste estudo, consideramos os jogos Cloak of Darkness (Roger Firth, 1999) e Búzios: Ecos da Liberdade (Comunidades Virtuais, 2010).

Este artigo está organizado da seguinte forma: a Seção 2 descreve conceitos importantes em jogos *adventure*; a Seção 3 explica os principais elementos da linguagem Machinations; a Seção 4 apresenta uma proposta de padrões para modelar elementos de jogos *adventure* com Machinations; a Seção 5 relata uma avaliação do uso desses padrões para modelar partes de jogos *adventure*; a Seção 6 descreve trabalhos relacionados; e a Seção 7 apresenta conclusões e trabalhos futuros.

2 JOGOS DE *adventure*

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

*e-mail: email@somewhere.com

Conceito	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 em jogos de *adventure*

forma, descrevemos o escopo de jogos de *adventure* com base nas características mais comuns, sumarizadas na Tabela 1.

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 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[1].

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 [9].

Os cenários são ambientes pelos quais o jogador passa durante o percorrer do jogo. Isso pode incluir salas, quartos, ambientes externos, entre outros.

Consideramos como objetos interativos os objetos com os quais o jogador pode realizar alguma interação. As interações podem ser acender uma lâmpada, abrir uma porta, 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' e acender a lâmpada alteraria a variável de um ambiente de 'escuro' para 'iluminado'.

Deixamos claro que NPCs, itens coletáveis e objetos interativos sempre estão situados em algum cenário. 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 *puzzles* 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 no projeto de 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 necessária para prosseguir sua aventura, ele não terá mais como avançar e nem retroceder. Obviamente é uma situação que deve ser evitada, caso contrário o jogador teria de reiniciar o jogo.

3 MACHINATIONS

Machinations foi concebido por Joris Dormans [8] para ajudar *game designers* a criar, representar e testar economias de jogos. A economia interna de um jogo é um sistema no qual seus elementos são produzidos, consumidos e trocados em quantidades quantificáveis. A maioria dos jogos tem uma economia interna, embora a complexidade e a importância da economia interna varie consideravelmente de gênero para gênero [1].

O Machinations foi proposto para, além de ser uma linguagem de modelagem visual de fácil compreensão para não-programadores, possibilitar executar os diagramas e validar mecânicas em tempo real através de uma ferramenta criada e disponibilizada online[2].

Diagramas em Machinations contêm diversos elementos, dentre os quais os principais são nós e conexões; uma conexão geralmente interliga dois nós (mas há exceções, como veremos a seguir). Nesta Seção explicaremos os tipos de nó e os tipos de conexão que fazem parte dos padrões de modelagem de jogos de *adventure*, apresentados na Seção 4. Em particular, serão explicados os nós do tipo *reservatório*, *portão*, *fonte*, *dreno* e *condições de fim*, e as conexões *de recurso* e *de estado*.

3.1 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. 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.

Na ciência da computação, um reservatório (*pool*) é 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 2. Quando a quantidade de recursos dentro de um reservatório é maior que 25, os recursos passam a ser representados de forma numérica ao invés de pequenos círculos.

Figura 2: Reservatórios.

3.2 Conexões de recursos

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 numéricas e até fórmulas para definir resultados aleatórios. Não tendo nada nesse campo, a taxa fica definida automaticamente como 1.

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, esses nós são representados como na Figura 3.

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

[2] Disponível em `http://www.jorisdormans.nl/machinations/`

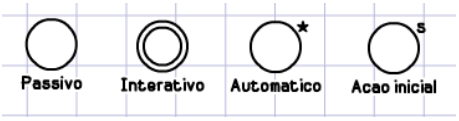

Figura 3: Modos de ativação.

Os modos de ativação se aplicam a quase todos os tipos de nó, exceto para nós de registro e de estado final, sendo que o primeiro não será abordado neste artigo.

Quando um nó é acionado, o comportamento de resposta do nó é determinado pelo seu **modo de transmissão**. Esse 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 4 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 4, temos como resultado da transmissão de recursos apresentado na Figura 5.

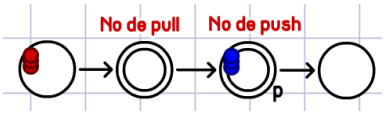

Figura 4: Modos de transmissão de recursos.

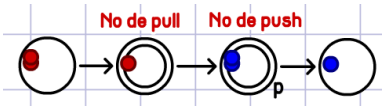

Figura 5: Transmissão de recursos após acionamento de nós.

Se a taxa de fluxo de uma conexão for x, mas o número de recursos a serem puxados ou empurrados através dessa conexão for y, sendo $y < x$, então podem ocorrer duas situações:

- se o modo *push/pull* do nó for ***push/pull any***, então esse nó irá empurrar ou puxar y recursos, mesmo y sendo uma quantidade menor do que a especificada x;
- se o modo *push/pull* do nó for ***push/pull all***, então esse nó não irá empurrar ou puxar nenhum recurso.

Note na Figura 6 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.

3.3 Conexões de estado

O estado de um diagrama do Machinations é definido pela distribuição de recursos entre seus nós. Mudanças de estado 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 linhas tracejadas e direcionadas, ligando o nó de origem a um alvo, que pode ser um nó ou o rótulo de 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, sendo os dois últimos definidos a seguir.

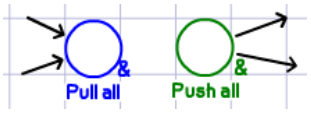

Figura 6: Nós de *push* e *pull all*.

3.3.1 Gatilhos

Como o nome já diz, os gatilhos (*triggers*) acionam um determinado elemento alvo do diagrama, seja ele um nó ou rótulo, quando todas as condições das conexões de entrada do nó de origem forem atendidas. Para indicar que uma conexão de estado é um gatilho, utiliza-se o símbolo asterisco '*' em seu rótulo. Se o alvo do gatilho for um nó, este alvo irá puxar ou empurrar recursos de acordo com seu modo, caso seja o rótulo de uma conexão de recursos, a conexão irá transmitir a quantidade de recursos indicada.

No exemplo da Figura 7 vemos que apenas quando **ambos** os reservatórios A e B transmitem seus recursos resulta no cumprimento das conexões de entrada do nó de origem C, acionando seu gatilho e resultando na transmissão de recurso do reservatório alvo D para o reservatório E.

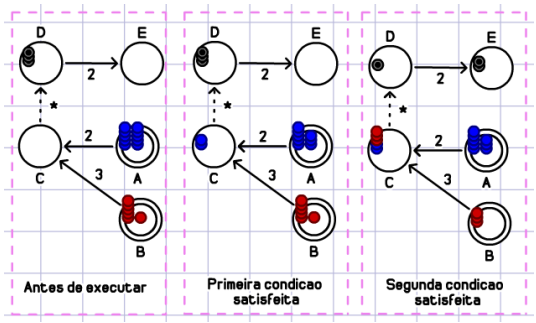

Figura 7: Exemplo de gatilho aplicado a um nó.

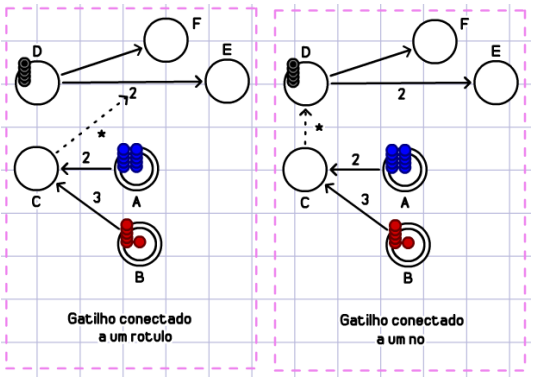

Figura 8: Exemplo de gatilho aplicado a um rótulo.

Utilizamos como base o diagrama da Figura 7 e modificamos para que o gatilho acionasse um rótulo. Note no primeiro exemplo da Figura 8 que quando o gatilho é acionado, apenas a conexão entre os reservatórios D e E transmite os recursos definidos por sua taxa de fluxo. Em contrapartida se o gatilho estivesse conectado ao reservatório D, como vemos no segundo exemplo da Figura 8, todas as conexões de saída do reservatório alvo transmitiriam os recursos de acordo com suas taxas.

3.3.2 Ativadores

Ativadores (*activators*) conectam nós e podem ativar ou suspender o nó alvo dependendo das condições especificadas em seus rótulos. Para cada ativador de um nó alvo, é verificada a validade da expressão lógica envolvendo os números de recursos de seu nó de origem. Se todas as condições forem satisfeitas, o nó alvo é ativado. Caso contrário, será suspenso.

Podemos ver no exemplo modelado na Figura 9 que, para transportar o recurso madeira para o nó de inventário do jogador, são necessários lenhadores (representados por recursos) no reservatório floresta. O gatilho (rótulo '*') faz com que, a cada 5 madeiras transportadas, um lenhador volte para seu reservatório de lenhadores. O ativador (rótulo '>0') faz com que o nó alvo, 'Madeiras', esteja ativo apenas quando o nó de origem, 'Floresta' possuir um número de recursos maior que zero; como resultado, quando não existe mais nenhum recurso de lenhador no reservatório de floresta, não há como movimentar mais as madeiras para o inventário do jogador.

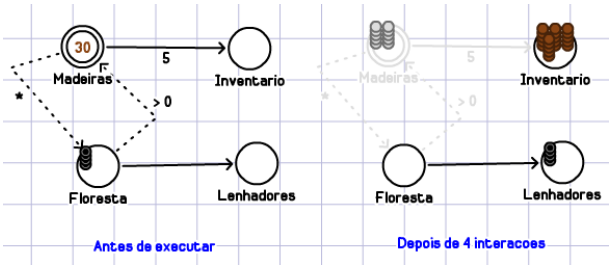

Figura 9: Exemplo de um ativador.

3.4 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 (descritas por expressões lógicas). Vale ressaltar que todas as conexões de saída de um portão devem ser do mesmo tipo. Neste trabalho vamos abordar apenas conexões com probabilidade, que podem ser de dois tipos:

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

$$\overline{p} = \frac{1}{n}\sum_{i=1}^{n} p_i \qquad (1)$$

  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% os recursos excedentes serão destruídos.

Portões possuem dois modos de distribuição, o determinístico e o aleatório. Um portão determinístico irá distribuir os recursos uniformemente de acordo com as probabilidades de distribuição e sua representação é dada por um losango. Já o portão aleatório irá gerar um valor entre 0% e 100% ou um valor abaixo da soma total dos pesos, e este nó é representado por um losango com um dado em seu interior.

Na Figura 10 temos exemplos de conexões com pesos e com porcentagens, além de portões de tipos determinístico e aleatório. Em seu exemplo 1, como o portão é determinístico o resultado será

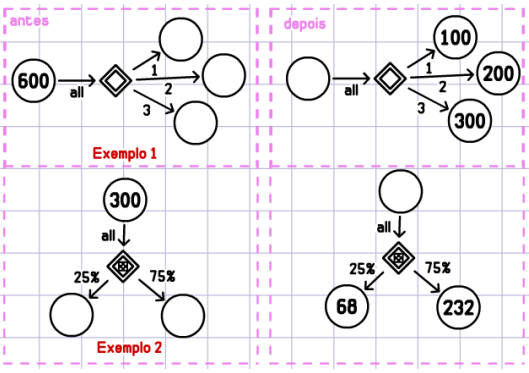

Figura 10: Exemplo de portões.

sempre o mesmo, sendo este determinado por seus pesos. Um reservatório terá $\frac{1}{6}$ * 600, outro terá $\frac{2}{6}$ * 600 e por fim o último terá $\frac{3}{6}$ * 600. No exemplo 2, como temos um portão do tipo aleatório, seus valores são gerados com probabilidades de 25% e 75%, portanto os resultados podem variar a cada execução. Caso esse nó fosse determinístico a distribuição de recursos nos reservatórios seria sempre de 75 e 225.

3.5 Fontes e drenos

Os nós fonte (*source*), representados por triângulos, geram recursos e os transmitem para um reservatório. Eles não possuem conexões de entrada, e podem gerar uma quantidade infinita de recursos.

Como oposto de uma fonte, os drenos (*drains*) são representados por triângulos invertidos e 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 11 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.

Figura 11: Exemplo de dreno e fonte.

3.6 Conversores

Modelado para converter recursos, os conversores (*converters*) consomem os recursos de entrada e criam os recursos de saída. Portanto, podem ser modelados também com drenos e fontes, como exemplificado na Figura 12.

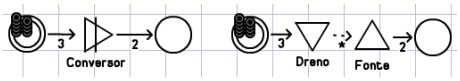

Figura 12: Conversor e sua representação com dreno e fonte.

3.7 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 13, cujo nó de fila transmite um recurso a cada três passos de tempo. Nesta mesma Figura temos um exemplo de geração de 5 tropas inimigas a cada 3 passos de tempo. Em adição foi utilizada uma fonte que gera recursos de forma automática enquanto o nó de atraso está em processamento, apenas com a função de contar quantos passos de tempo já se passaram.

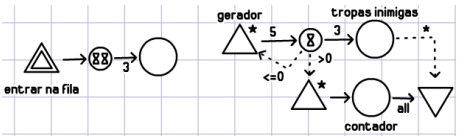

Figura 13: Exemplos de nós de atraso e fila.

3.8 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. Uma condição de fim é representada por um quadrado contendo um quadrado menor, preenchido, e deve ser ativada por ativadores.

Como representado na Figura 14, o jogador vence o jogo ao alimentar seu gato que estava faminto. Para isso, é necessário comprar comida através de um nó de portão que puxará o recurso gerado por uma fonte, acionando também o reservatório de dinheiro para destruir seu recurso de compra. Após realizar a compra, o jogador poderá alimentar seu gato deixando-o satisfeito e tornando a condição de vitória verdadeira, o que ocasiona no término da execução do diagrama.

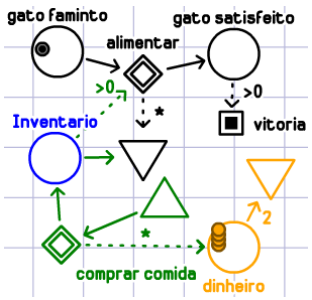

Figura 14: Estados finais.

3.9 Cores e elementos ilustrativos

Por padrão, as cores em um diagrama Machinations possuem apenas função estética; esse comportamento pode ser alterado através da opção *color coding*. Caso essa opção seja ativada, uma conexão de recursos transmitirá apenas recursos da mesma cor que ela, assim como conexões de estado validarão suas respectivas condições relacionadas a recursos de cor igual. Essa propriedade pode ser bem útil, visto que recursos com cores diferentes podem se agrupar em um mesmo reservatório.

Além dos nós e conexões explicados anteriormente, o editor de Machinations permite adicionar a um diagrama elementos com função ilustrativa: *TextLabel* (texto) e *GroupBox* (caixa tracejada com título).

4 Padrões de modelagem de jogos de *adventure* com Machinations

Tendo visto uma breve introdução sobre as características básicas de jogos de *adventure* na Seção 2 e os conceitos básicos utilizados no Machinations na Seção 3, podemos agora propor uma abordagem na modelagem de jogos de *adventure* utilizando este *framework*.

4.1 Estruturas

4.1.1 Jogador e cenários

O personagem principal deverá sempre estar em um cenário. Por isso, ele será considerado um recurso que transitará entre reservatórios que representam cenários. Vamos definir uma cor única para o recurso que representa o jogador, sendo esta definida como a cor preta nos exemplos seguintes.

Na Figura 15 existem duas soluções representando a movimentação de um jogador entre quatro cenários que, neste caso, são salas. A sala 2 tem acesso às outras três salas, e as salas 1, 3 e 4 têm 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 essas saídas conexões de recursos que se ligam a um outro cenário.

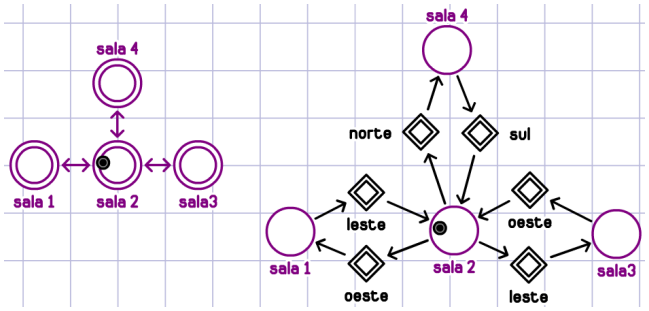

Figura 15: Representação do jogador e de cenários.

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 através de cores diferentes, portanto a opção de *Color Coding* da ferramenta Machinations deve ser marcada.

Na Figura 16 os itens são representados por recursos de cor marrom e azul, sendo estes situados no reservatório 'banheiro'.

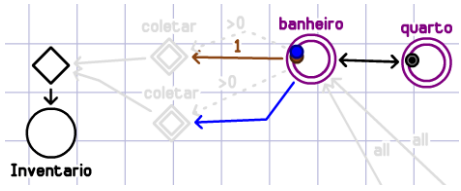

Figura 16: Representação de itens coletáveis.

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 cada ação de coleta;
- entre o reservatório do cenário e os nós de portão de coleta haverá conexões de estado com a 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, entre seu nó de portão de coleta 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.
- todos os nós de coleta serão ligados a um portão que transmitirá os itens ao inventário, desta forma diminuindo a quantidade de conexões de entrada no mesmo.

Na Figura 16 o jogador ainda não chegou ao nó que representa o banheiro onde se encontram os itens coletáveis que ele precisa, portanto a coleta destes itens ainda não está disponível. Após chegar no banheiro, exemplificado na Figura 17, as conexões de estado entre o banheiro e seus nós de coleta são atendidas, habilitando a ação de coletar.

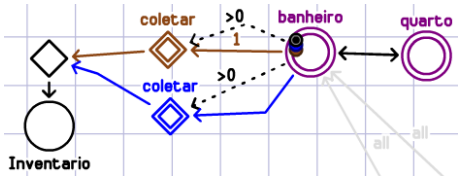

Figura 17: Coleta de itens habilitada após jogador alcançar a sala requerida.

4.1.3 Distribuição inicial de itens

Ao criarmos o diagrama sobre a coleta de itens representado na Figura 17, percebeu-se que nós de reservatórios não têm a opção de definir recursos de cores distintas em sua composição inicial. Dessa forma, surgiu a necessidade da criação do padrão **fábrica de itens**. A fábrica de itens tem como objetivo distribuir recursos distintos em reservatórios no início da execução do diagrama. Observe na Figura 18 como funciona sua representação. A GroupBox é opcional, mas é recomendada para melhor visualização. Dentro dela, são colocados nós com modo de ativação inicial, ligados através de conexões de recurso aos seus reservatórios-alvo, realizando desta forma a distribuição de itens no início do jogo. O rótulo para as conexões de recurso é especificado como 'all'.

4.1.4 Variáveis, ações e interações com objetos

Ações que envolvam variáveis do sistema ou condições para que uma ação seja tomada são modeladas com nós de portão. Esses 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 Seção 4.1.2 utilizamos um nó de portão para representar a ação de coleta de itens. Na Figura 19 observamos que o nó que representa a ação de "sair" é representado por um nó de portão, cuja ativação apenas ocorre quando a condição de estar com a carteira no inventário é atendida, visto que o jogador vai ao mercado.

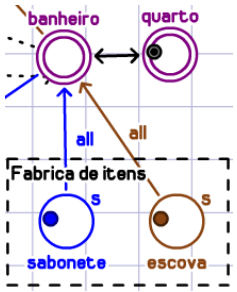

Figura 18: Fábrica de itens.

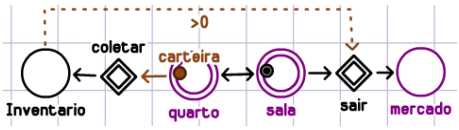

Figura 19: Representação de ações.

4.1.5 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 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.

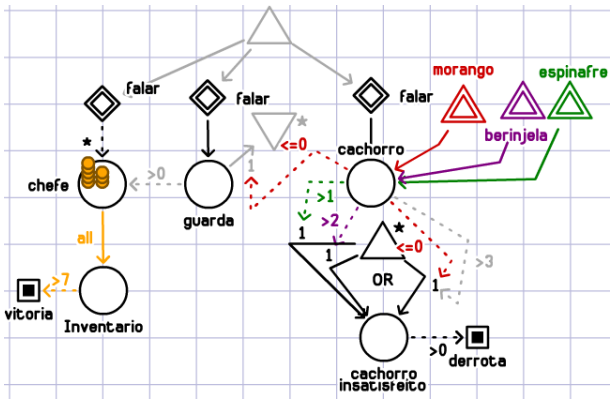

Figura 20: Representação de interações com NPCs.

No exemplo da Figura 20 o repositório do NPC chefe guarda recursos de dinheiro. O objetivo do jogo é obter esse dinheiro para o inventário do jogador. 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 essa 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. Os recursos relacionados com as comidas são gerados por nós fonte. Outro ponto importante a ser observado foi a necessidade da definição de uma disjunção lógica. Este conceito é melhor definido na seguinte Seção 4.1.6.

4.1.6 Conectivos lógicos

Antes de explicar conjunção e disjunção lógica, precisamos de dois conceitos básicos: proposições e conectivos lógicos. Uma proposição é uma sentença declarativa (isto é, uma sentença que declara um fato) que é verdadeira ou falsa, mas não ambas [11]. Já conectivos lógicos são operadores lógicos que são usados para formar novas proposições a partir de duas ou mais proposições existentes.

Dessa forma, as definições de **conjunção** e **disjunção lógicas** seguem da seguinte forma:

- Sejam 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 20. 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 **> 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 21 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.

4.1.7 *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. 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* e ritmo musical. 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*). Basta especificar as probabilidades de ganhar ou perder o *minigame* através de conexões de recursos. Na Figura 22 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*.

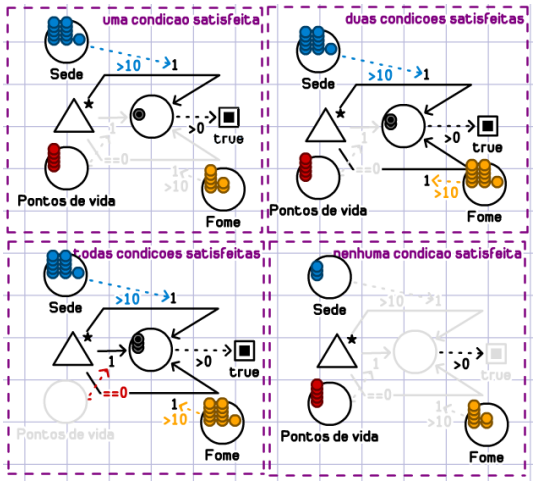

Figura 21: Representação de uma disjunção lógica.

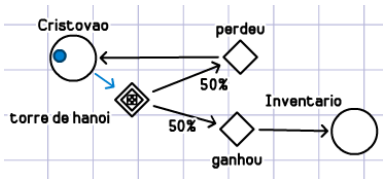

Figura 22: Nó não-determinístico aleatório.

4.2 *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 23 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 de custo muito alto dentro do escopo do Machinations.

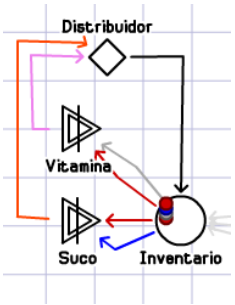

Figura 23: Conversão de recursos.

4.3 Mecanismo de *lock and key*

Com as estruturas anteriores definidas, poderíamos modelar um simples mecanismo chamado de *lock-and-key*. Nesse mecanismo,

o progresso da experiência é bloqueado (a fechadura ou *lock*) até que o jogador realize a tarefa desejada (a chave ou *key*) [10].

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.

A modelagem do mecanismo *lock-and-key* é exemplificada na Figura 24. Para que o jogador possa abrir a porta no Quarto 2, é necessário ir para o Quarto 1 coletar a chave necessária. Estando no quarto onde a porta se encontra, e tendo a chave em seu inventário, a interação de abrir a porta é habilitada dessa forma possibilitando o avanço do jogador.

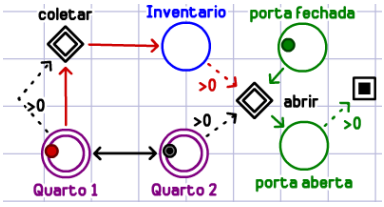

Figura 24: Exemplo de diagrama utilizando o mecanismo de *lock-and-key*.

5 Avaliação

Nesta Seção mostraremos a aplicabilidade dos padrões introduzidos na Seção 4 para a modelagem de jogos de *adventure*. Para isso, utilizaremos três estudos de caso:

- jogo das 3 salas, projetado para exemplificar o mecanismo de lock-and-key;
- Cloak of Darkness, IF criada por Roger Firth[3] para comparar sistemas de criação de IFs;
- Búzios: Ecos da Liberade, jogo de *adventure* 2D *point & click* desenvolvido pelo grupo de pesquisa Comunidades Virtuais [6] em 2010 sobre a Guerra dos Búzios.

5.1 Estudo de caso 1: jogo das 3 salas

Trazemos com este estudo de caso um exemplo simplificado de aplicação do mecanismo de *lock-and-key* mencionado na Seção 4.3. O diagrama está representado na Figura 25 e 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.

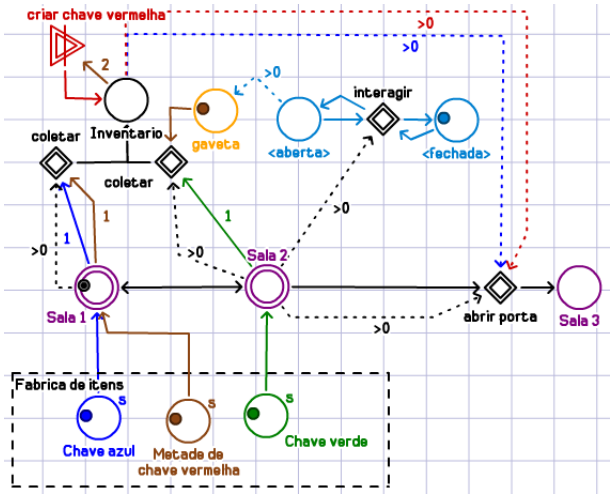

Figura 25: Jogo das 3 salas utilizando o mecanismo de *lock-and-key*.

As chaves são itens coletáveis de cenário, modeladas com cores diferenciadas. O padrão de coleta na sala 1 foi simplificado para que os itens fossem coletados de vez. Para a criação da chave vermelha a partir das metades de chave vermelha, foi utilizado o padrão de *crafting* da Seção 4.2. A distribuição dos itens no cenário foi feita através do padrão da fábrica de itens.

A gaveta e a porta são objetos interativos modelados de formas diferentes. Como neste caso o objetivo final do jogo é chegar à 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 interativo 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.

5.2 Estudo de caso 2: Cloak of Darkness

A especificação do Cloak of Darkness é descrita 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

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

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.

5.2.1 Modelagem utilizando Machinations

Os padrões utilizados foram basicamente os de movimentação do personagem, descrito na Seção 4.1.1, e do uso de portões para representação de ações, descrito na Seção 4.1.4. Como as informações de localização dos cenários são descritas na especificação do jogo, surgiu a necessidade de modelar ações de movimentação direcionais entre esses cenários. Os cenários foram modelados com reservatórios de cor azul e o manto como um recurso verde iniciando no inventário. Esse 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 maior do que a especificada, o jogador perde o jogo. Essa modelagem se encontra na Figura 26.

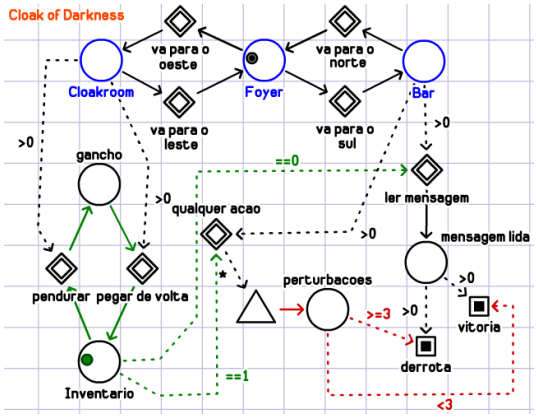

Figura 26: Cloak of Darkness no Machinations.

5.3 Estudo de caso 3: fase do convés do jogo Búzios

Existem duas características diferentes do exemplo da Seção 5.1 que valem ser mencionadas: (a) toda a fase se passa em um único cenário, não existe a necessidade de modelar onde o jogador se encontra fisicamente; (b) 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 Seção 4.1.5, 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 Seção 4.1.4 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 conversor e um nó de atraso. O nó conversor veio do padrão de *crafting*, que foi utilizado neste caso 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. 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 28.

Figura 27: Cena do convés com o avatar Francisco e NPCs.

A Figura 27 mostra uma captura de tela da fase do convés, estando presentes nela os personagens Roderico, Cristovão e o jogador principal Franciso ao centro.

6 Trabalhos Relacionados

Taylor et al. demonstraram 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 profissionais que não sejam da área de computação, como roteiristas, *game designers* ou artistas [12]. Cook [7] 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 [13]. Araújo e Roque descrevem uma abordagem com redes de Petri para modelar jogos [3]; nos jogos Europe 2045 [5] e Karo [4], as redes de Petri foram utilizadas para modelagem na fase de especificação.

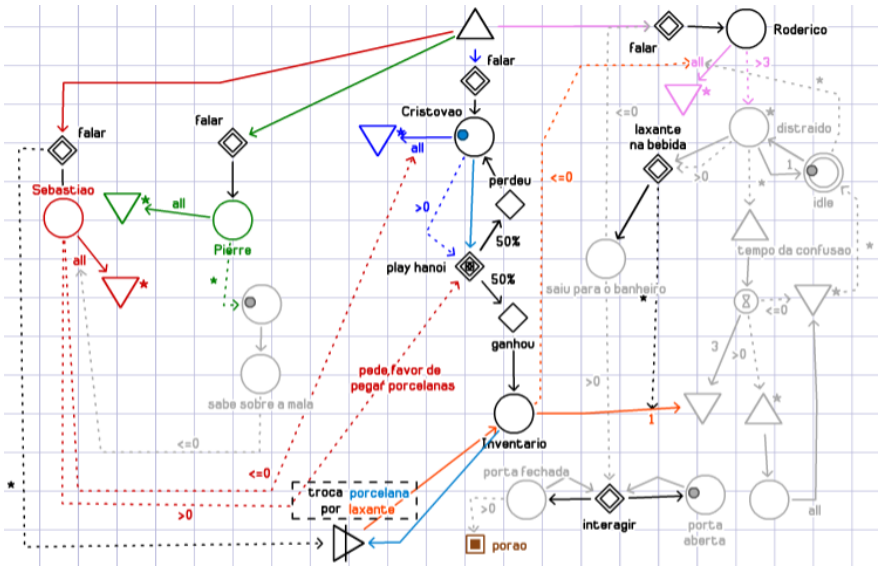

Figura 28: Diagrama completo da fase do convés do jogo Búzios.

XXX e YYY[4] usaram redes de Petri coloridas (CPNs, do inglês *colored Petri nets*), uma extensão de redes de Petri, para modelar, simular e verificar formalmente propriedades de jogos de *adventure*, usando como exemplo o jogo Cloak of Darkness [14]. O modelo do jogo elaborado em CPN por eles possui 12 nós e 29 conexões (chamadas de arcos), contra 18 nós e 25 conexões do modelo em Machinations proposto neste trabalho, o que sugere que os dois formalismos possuem expressividade similar. Diferentemente dos modelos em Machinations, as CPNs possuem apenas dois tipos de nó (lugares e transições); por outro lado, suas conexões são mais complexas, incluindo em alguns casos rótulos com expressões escritas em uma linguagem de programação.

7 Conclusão

Neste trabalho realizamos a proposta de mapeamento de jogos de *adventure* na linguagem visual Machinations, obtendo a modelagem de dois jogos selecionados. Como principal contribuição temos a criação de padrões para a modelagem de jogos de *adventure*, nos levando a conclusão de que é factível o uso do Machinations para modelar jogos do gênero, mesmo não sendo este o foco da linguagem por não ter uma economia interna.

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 para desenvolver uma linguagem que atenda de forma ampla 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.

Referências

[1] E. Adams. *Fundamentals of Game Design*. New Riders Publishing, Thousand Oaks, CA, USA, 2nd edition, 2009.

[2] E. Adams and J. Dormans. *Game Mechanics: Advanced Game Design*. New Riders Publishing, Thousand Oaks, CA, USA, 1st edition, 2012.

[3] M. Araújo and L. Roque. Modeling Games with Petri Nets. In B. Atkins, H. Kennedy, and T. Krzywinska, editors, *Breaking New Ground: Innovation in Games, Play, Practice and Theory: Proceedings of the 2009 Digital Games Research Association Conference*, London, Sept. 2009. Brunel University.

[4] D. Balas, C. Brom, A. Abonyi, and J. Gemrot. Hierarchical petri nets for story plots featuring virtual humans. In *Proceedings of the Fourth AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, AIIDE'08, pages 2–9. AAAI Press, 2008.

[5] C. Brom, V. Sisler, and T. Holan. Story manager in 'europe 2045' uses petri nets. In *Proceedings of the 4th International Conference on Virtual Storytelling: Using Virtual Reality Technologies for Storytelling*, ICVS'07, pages 38–50, Berlin, Heidelberg, 2007. Springer-Verlag.

[6] Comunidades Virtuais. Búzios: Ecos da liberdade, 2011.

[7] D. Cook. The chemistry of game design, 7 2007. Article in the Gamasutra webzine.

[8] J. Dormans. Machinations: Elemental Feedback Patterns for Game Design. In J. Saur and M. Loper, editors, *GAME-ON-NA 2009: 5th International North American Conference on Intelligent Games and Simulation*, pages 33–40, 2009.

[9] E. Ju and C. Wagner. Personal computer adventure games: Their structure, principles, and applicability for training. *SIGMIS Database*, 28(2):78–92, Apr. 1997.

[10] D. L. Roberts, M. L. Furst, B. Dorn, and C. L. Isbell. Using influence and persuasion to shape player experiences. In *Proceedings of the 2009 ACM SIGGRAPH Symposium on Video Games*, Sandbox '09, pages 23–30, New York, NY, USA, 2009. ACM.

[11] K. H. Rosen. *Discrete mathematics and its applications*, volume 10. 2007.

[12] M. J. Taylor, D. Gresty, and M. Baskett. Computer game-flow design. *Comput. Entertain.*, 4(1), Jan. 2006.

[13] C. Verbrugge and P. Zhang. Analyzing computer game narratives. In *Entertainment Computing – ICEC 2010, 9th International Conference*, number 6243 in LNCS, pages 224–231. Springer-Verlag, September 2010.

[14] XXXXX and XXXXX. Xxxxx. In *Brazilian Symposium on Computer Games and Digital Entertainment (in press)*, 2017.

[15] T. Zaidan, D. Zaidan, and L. F. W. Góes. Game mechanics design: Applying machinations to eliosi's hunt. In *Brazilian Symposium on Computer Games and Digital Entertainment (SBGAMES) - Art & Design Track*, pages 313–321, São Paulo, Brazil, 2016. SBC.

[4]Nomes omitidos por causa da avaliação *double-blind*.