PEGASUS: UMA FERRAMENTA DE SIMULAÇÃO PARA APOIO AO DESIGN DE JOGOS DE PROGRESSÃO

Marcelo Arêas Rodrigues da Silva

Tese de Doutorado apresentada ao Programa de Pós-graduação em Engenharia de Sistemas e Computação, COPPE, da Universidade Federal do Rio de Janeiro, como parte dos requisitos necessários à obtenção do título de Doutor em Engenharia de Sistemas e Computação.

Orientador: Geraldo Bonorino Xexéo

Rio de Janeiro

Março de 2018

PEGASUS: UMA FERRAMENTA DE SIMULAÇÃO PARA APOIO AO DESIGN DE JOGOS DE PROGRESSÃO

Marcelo Arêas Rodrigues da Silva

TESE SUBMETIDA AO CORPO DOCENTE DO INSTITUTO ALBERTO LUIZ COIMBRA DE PÓS-GRADUAÇÃO E PESQUISA DE ENGENHARIA (COPPE) DA UNIVERSIDADE FEDERAL DO RIO DE JANEIRO COMO PARTE DOS REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE DOUTOR EM CIÊNCIAS EM ENGENHARIA DE SISTEMAS E COMPUTAÇÃO.

Examinada por:

_____

Profº Geraldo Bonorino Xexéo, D.Sc.

_____

Profº Jano Moreira de Souza, Ph.D.

_____

Profº Renata Mendes de Araujo, D.Sc.

_____

Profº Adriana Santarosa Vivacqua, D.Sc.

_____

Profª Carlos Eduardo Ribeiro de Mello, D.Sc.

RIO DE JANEIRO, RJ – BRASIL

MARÇO DE 2018

Silva, Marcelo Arêas Rodrigues da

Pegasus: Uma Ferramenta de Simulação para Apoio ao Design de Jogos de Progressão / Marcelo Arêas Rodrigues da Silva. – Rio de Janeiro: UFRJ/COPPE, 2018.

XV, 153 p.: il.; 29,7 cm.

Orientador: Geraldo Bonorino Xexéo

Tese (doutorado) – UFRJ / COPPE / Programa de Engenharia de Sistemas e Computação, 2018.

Referências Bibliográficas: p. 113 - 121

1. Design de jogos 2. Simulação I. Xexéo, Geraldo Bonorino. II. Universidade Federal do Rio de Janeiro, COPPE, Programa de Engenharia de Sistemas e Computação. III. Título.

À minha esposa Emanuelle, meus pais e avós, por todo amor, carinho e incentivo de sempre.

AGRADECIMENTOS

A Deus, pelo dom da vida a cada dia, pela força e paciência necessárias em toda minha caminhada.

À minha esposa Emanuelle, por todo amor, incentivo e compreensão. Começamos a namorar no mesmo mês que ingressei no Doutorado, agora este capítulo chega ao fim, mas nossa história ainda tem muitas páginas para escrevermos juntos.

Aos meus pais Mara e Célio, meus avôs e avós, vivos e falecidos, Gelcy, Joaquim, Dalva e Romeu. Obrigado por toda a minha criação, pelos valores passados, por cada ensinamento, por todo apoio e até mesmo por cada bronca que levei, certamente com razão.

Ao meu irmão Raphael e meu cunhado Yago, pelas discussões produtivas ou não, e por todos os momentos de descontração e jogatinas.

Ao orientador e amigo Geraldo Xexéo, por rejeitar várias ideias e sugestões, até chegarmos ao tema dessa tese. Por sua paciência, seus conselhos, companhia nos almoços, pelos momentos que demonstrou preocupação, mas nunca falta de confiança. Sua orientação cirúrgica foi fundamental para a conclusão deste trabalho.

Aos professores Jano Moreira de Souza, Renata Araujo, Adriana Vivacqua e Carlos Eduardo Mello, por aceitarem participar da banca examinadora e emprestarem seus conhecimentos para agregar valor à avaliação deste trabalho.

Aos companheiros de doutorado e projetos, Fabrício Pereira e Eduardo Mangeli, pela amizade, cumplicidade, horas e mais horas de discussões sobre qualquer assunto.

A todos os membros do LUDES, tanto o grupo de pesquisa, quanto os presentes fisicamente no laboratório, meus agradecimentos por todos os momentos que passamos juntos nesses últimos anos.

A todos com quem tive a oportunidade de trabalhar e conviver durante o período na GPE, na primeira metade dessa caminhada. Obrigado pelas oportunidades, por todo o conhecimento compartilhado e os lanches com o famoso cafezinho.

À CAPES, pelo apoio financeiro, à COPPETEC pelo crescimento profissional e à COPPE/PESC pela excelência de ensino. É uma satisfação e motivo de orgulho poder bater no peito e dizer que a UFRJ é responsável por todo meu ensino superior e pós-graduação.

À Ana Paula, Patrícia, Solange, Guty e toda a equipe da secretaria, pela boa vontade e apoio que sempre demonstraram comigo e outros alunos durante todos esses anos de convivência acadêmica.

Aos meus amigos desde a infância e adolescência, Fernando, Rafael, João, Mi, Bia, Adriano, Dani, Francine, Windson. Uns mais pertos, outros mais longe, e uma no céu olhando por todos nós. Obrigado pela amizade.

Aos meus padrinhos e madrinhas, de batismo e de casamento, Carla, Jorge, Paula, Rodrigo, Ester, Marcello, Ana Luiza, Marcio, pela amizade e a certeza em saber que sempre posso contar com vocês para qualquer coisa.

Ao grupo de carona SG – Fundão, que tornaram minhas idas e vindas à universidade muito mais prazerosas. Obrigado pela amizade e horas de conversas no carro. Juntos, nós rimos na cara dos engarrafamentos.

Por fim, agradeço a todos que me ajudaram de alguma forma na realização deste trabalho. Chegar até aqui não é fácil, mas garanto que vale a pena. Que venham novos desafios e a todos vocês o meu eterno MUITO OBRIGADO!

Resumo da Tese apresentada à COPPE/UFRJ como parte dos requisitos necessários para a obtenção do grau de Doutor em Ciências (D.Sc.)

PEGASUS: UMA FERRAMENTA DE SIMULAÇÃO PARA APOIO AO DESIGN DE JOGOS DE PROGRESSÃO

Marcelo Arêas Rodrigues da Silva

Março / 2018

Orientador: Geraldo Bonorino Xexéo

Programa: Engenharia de Sistemas e Computação

A indústria de jogos tem apresentado grande crescimento e os números demonstram que ela continua a se expandir. Simultaneamente, a popularização dos *smartphones* contribui com a crescente inclusão de jogos no cotidiano. O processo de construção de um jogo é composto de uma série de etapas, sendo o *design* uma das etapas iniciais. O *designer* de jogo tem como desafio transportar a ideia de sua mente para o jogo de maneira satisfatória. Entretanto, não existem muitas ferramentas especializadas que auxiliem esse trabalho.

Nessa tese é proposto um ambiente para apoiar o *design* de jogos com estrutura de progressão, por meio de uma ferramenta de simulação. A proposta descreve a interação do *designer* de jogo com o simulador para que este possa auxiliar no processo de criação, teste e aperfeiçoamento dos elementos de jogo antes de prosseguir para a etapa de programação.

Para cumprir esse propósito foi construído o simulador de jogos de progressão Pegasus, baseado em padrões de desenvolvimento de *software* e padrões para *design* de jogos. São apresentados sua arquitetura, funcionamento e a experiência de utilização para apoiar o *design* de alguns tipos de jogos. Por fim, são expostas as conclusões e possibilidades de expansão da ferramenta.

Abstract of Thesis presented to COPPE/UFRJ as a partial fulfillment of the requirements for the degree of Doctor of Science (D.Sc.)

PEGASUS: A SIMULATION TOOL TO SUPPORT DESIGN OF PROGRESSION GAMES

Marcelo Arêas Rodrigues da Silva

March / 2018

Advisor: Geraldo Bonorino Xexéo

Department: Systems and Computing Engineering

The video game industry has shown great growth and the numbers show that it continues to expand. Simultaneously, the popularization of smartphones contributes to the growing inclusion of games in daily life. The process of building a game consists of a series of steps, the design being one of the initial stages. The game designer has the challenge of carrying the idea of his mind to the game in a satisfactory way. However, there are not many specialized tools to support this work.

This thesis proposes an environment to support the design of games with progression structure, through a simulation tool. The proposal describes the interaction of the game designer with the simulator so that it can assist in the process of creating, testing and refining game elements before proceeding to the programming stage.

To fulfill this purpose Pegasus progression game simulator was built, based on patterns for software development and patterns for game design. This thesis presents its architecture, operation and the experience using the tool to support the design of some games of different types. Finally, the conclusions and possibilities of tool expansion are exposed.

SUMÁRIO

LISTA DE FIGURAS

LISTA DE TABELAS

1. Introdução

As pesquisas para a presente tese demonstram que a indústria de jogos digitais vem apresentando expansão significativa ao longo dos últimos anos. No entanto, abordagens práticas e mais bem fundamentadas para criar bons jogos, ou mesmo avaliar alguns de seus aspectos ainda são insuficientes.

Neste capítulo introdutório são apresentadas as seguintes seções: a motivação, a definição do problema, a hipótese da proposta de pesquisa, o objetivo deste trabalho e a organização do texto.

1.1 Motivação

A evolução dos consoles, computadores e *smartphones* contribuem para o surgimento de jogos cada vez mais complexos. A indústria de jogos tem apresentado um grande crescimento. Uma pesquisa realizada anualmente nos Estados Unidos indica que, em 2016, havia pelo menos uma pessoa que joga videogame regularmente em 65% dos lares americanos (ENTERTAINMENT SOFTWARE ASSOCIATON, 2017). É considerado um jogador regular aquele que joga três horas ou mais por semana. A mesma pesquisa revela que existe uma média de 1,7 jogadores regulares em cada um desses lares.

Além da expansão no mercado, esse seguimento apresenta crescente movimentação financeira a cada ano. A Figura 1 apresenta o gasto anual, em bilhões de dólares, em jogos de videogame no período de 2010 a 2016. Somente em 2017 foram 24,5 bilhões de dólares gastos pelos americanos em jogos (ENTERTAINMENT SOFTWARE ASSOCIATON, 2017).

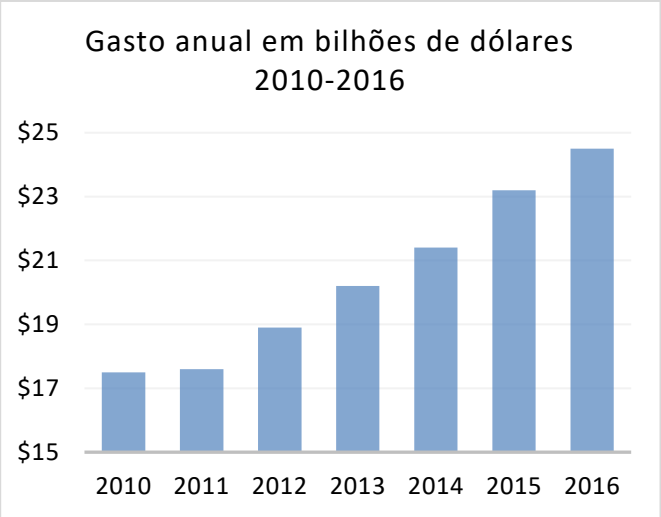

Figura 1: Gasto com videogame nos últimos anos. Adaptado de (ENTERTAINMENT SOFTWARE ASSOCIATON, 2017)

Considerando os gastos com aparelhos e acessórios, além dos jogos de videogame, esse número salta para 30,4 bilhões. A Figura 2 ilustra esse cenário.

Figura 2: Total gasto em 2017 na indústria de videogames. Adaptado de (ENTERTAINMENT SOFTWARE ASSOCIATON, 2017)

Em 2016, a média de idade do jogador americano foi de 35 anos. A pesquisa indica que o videogame está constituindo-se como atividade cultural do cidadão, ocupando parte do tempo que seria dedicado a assistir televisão ou ir ao cinema, por exemplo. Existem 26% de jogadores com idade a partir de 50 anos, o que faz uma boa contrapartida aos 29% de jogadores que possuem menos de 18 anos (ENTERTAINMENT SOFTWARE ASSOCIATON, 2017). Isso nos mostra que a premissa “videogame é atividade para crianças” é uma falácia. A Figura 3 ilustra esse cenário.

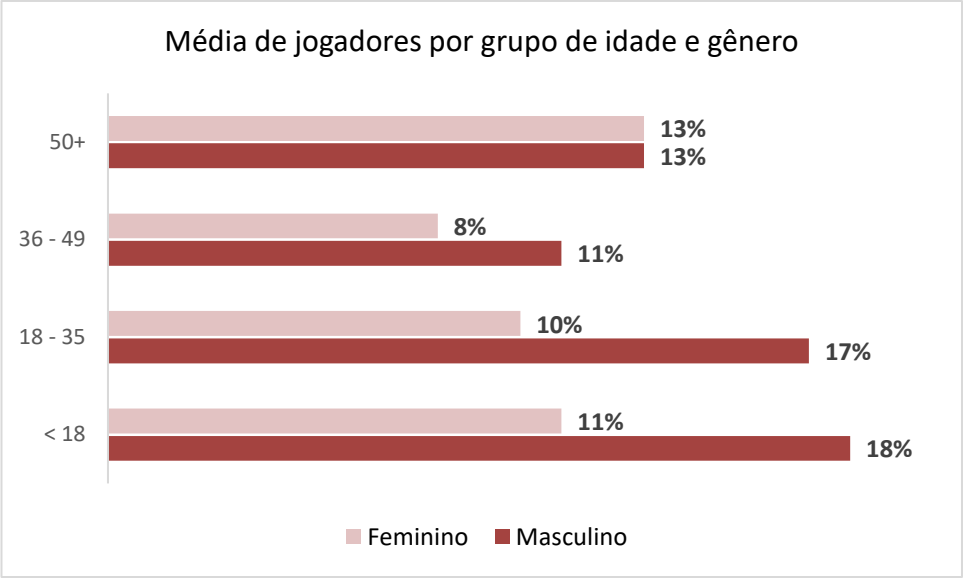

Figura 3: Jogadores por grupo de idade e gênero. Adaptado de (ENTERTAINMENT SOFTWARE ASSOCIATON, 2017)

O panorama no Brasil é mais discreto que nos Estados Unidos. Em 2017, o BNDES realizou uma pesquisa com empresas brasileiras do setor de jogos digitais e foi registrado um faturamento de 1,3 bilhões de dólares com perspectiva de crescimento de 7,3% ao ano (BNDES, 2018).

A maioria das empresas deste setor residentes no Brasil são pequenas e estão localizadas na região sul e sudeste do país, sendo pouco mais de 70% constituídas de até 5 colaboradores. No entanto, mais de 50% das empresas atuam tanto no mercado brasileiro quanto internacional (BNDES, 2018).

Os dados apresentados se restringem apenas ao universo dos videogames e jogos de computador. Além destes, ainda existem jogos de tabuleiro, jogos de cartas, jogos de azar, jogos com bola e outros. A partir desses dados, é razoável afirmar que as atividades lúdicas despertam interesse da população em geral.

1.2 Definição do problema

A criação de um jogo é um processo tão complexo quanto a criação de um sistema de informação. Existem diversos pontos a serem considerados, desde a concepção da ideia inicial, passando pelo *design*, implementação, testes, refinamento, até a obtenção do produto final. Na literatura, diversos autores tratam deste assunto (BETHKE, 2003)

(ROLLINGS & MORRIS, 2004) (CRAWFORD, 1984) (SCHELL, 2008) (SALEN & ZIMMERMAN, 2004) (ADAMS, 2009).

Entre as etapas de *design* e implementação podem ocorrer problemas de comunicação entre os membros dessas equipes capazes de comprometer a construção do jogo. Existem ferramentas suficientes para auxiliar o *designer* nesse processo? Como ele pode expressar suas ideias e como testá-las? Elas estão prontas para serem implementadas?

Raph Koster, em sua apresentação na *Game Developers Conference* de 2005 (KOSTER, 2005a), compara jogos a uma dança, onde "muitas disciplinas estão envolvidas, mas uma é crítica: coreografia". Música pode ser representada por meio de partitura e seus símbolos próprios, as notas musicais. Por meio desse sistema de padronização, uma música consegue ser reproduzida da mesma forma mundialmente. A pronúncia de palavras pode ser representada através do alfabeto fonético internacional, um sistema de notação fonética baseado no alfabeto latino como forma de representação padronizada dos sons.

Com a coreografia, contudo, não é tão simples. Até existe um sistema de notação chamado *Labanotation* (COHEN & HUTCHINSON, 1954) para representar o movimento do corpo humano que pode ser usado como notação para dança. Entretanto, trata-se de uma representação abstrata que não é utilizada em larga escala. Com os jogos ocorre algo similar, não há uma forma padronizada e eficaz de representação.

Durante as pesquisas para a tese foi identificado que não existe uma ferramenta única e completa capaz de fornecer apoio a todo o processo de construção de um jogo, possivelmente devido a limitações tecnológicas. Na verdade, conforme afirmam (NEIL, 2012) e (KOSTER, 2012), percebe-se uma carência de elementos para apoiar o *design* de jogos. Logo, é desejável que existam ferramentas especializadas que possam apoiar o *design* da melhor maneira possível.

Um dos pontos mais importantes e que pode ser decisivo para o sucesso ou fracasso de um jogo é o seu balanceamento. Assim como um sistema de informação, que precisa funcionar e não conter erros críticos, um jogo também precisa ser divertido. Um jogo que não consegue entreter o jogador está fadado ao fracasso. Analogamente, seria como um sistema de informação que foi construído, mas que não será utilizado. Um consenso entre os *designers* de jogo é que um jogo desbalanceado irá provocar rapidamente nos jogadores a perda de interesse (BURGUN, 2011).

Assim como existem diferentes processos de desenvolvimento de *software*, cada um sendo mais apropriado para um caso específico, os jogos também possuem diversos gêneros e estilos, que requerem variados modelos de desenvolvimento (SANTOS *et al.*, 2012).

Existem duas estruturas básicas de jogos em que podemos encaixar os jogos digitais, segundo Juul. São elas emergência e progressão (JUUL, 2002). Jogos com características de emergência são especificados como um pequeno conjunto de regras que combinam e produzem grandes quantidades de variações de jogo, como *Gamão* e *Tetris*. Jogos com progressão são caracterizados por possuírem uma sequência bem controlada de eventos, com desafios que o jogador encontra em cada etapa ou *level* (JUUL, 2002). O jogador precisa completar algum objetivo ou chegar ao final da fase para poder avançar no jogo, como em *Sonic* e *Alex Kidd*.

Em relação aos jogos de emergência, há o *framework Machinations* (DORMANS, 2012b) que pode auxiliar o *designer* em seu processo. Após utilizar o *framework* para criar o diagrama de representação de um jogo, seja ele simples ou complexo, o *Machinations* permite definir jogadores artificiais e rodar múltiplas execuções de um jogo, informando a porcentagem de vitória de cada jogador após o número desejado de execuções em sequência do jogo modelado. A seção 2.8.2 fornece mais detalhes sobre o *Machinations*.

Com o *Machinations*, a execução de simulações se mostrou um forte aliado para a observação do comportamento de um jogo a partir do seu *design* e, ainda, possibilita testes de diferentes cenários auxiliando no balanceamento de recursos.

Em relação aos jogos de progressão, não há ferramentas especializadas consolidadas que possam apoiar o *designer* na elaboração de jogos com essa estrutura. Portanto, há uma necessidade de pesquisas e novos trabalhos nessa área de forma a avançar o estado da arte, dado que os *designers* teriam muito a se beneficiar. Esta tese atua nesta frente de pesquisa e propõe um ambiente para simular jogos de progressão, com o objetivo de auxiliar o balanceamento de jogos e estreitar os laços entre as equipes de *designer* e implementação.

1.3 Questões de pesquisa

Existem ferramentas que dão suporte, isoladamente, a algumas etapas da construção de um jogo. No entanto apoiar a etapa de *design*, com possibilidade de realizar

simulações e obter algum resultado que auxilie o refinamento durante o *design* é algo ainda não consolidado em todo esse processo. Se o *designer* puder testar cenários e refinar o jogo durante sua concepção, antes da fase de implementação, é provável que o processo de construção de jogo seja beneficiado. Surge assim a ideia central dessa tese de se obter um ambiente para modelagem e simulação de jogos, para apoiar o *designer*

Durante as pesquisas para tese, um dos estudos realizados foi de abordagens para construção de modelos que representem jogos. Foi verificado que com Rede de Petri seria possível criar representações de jogos de progressão.

Inicialmente o trabalho com Rede de Petri, mais especificamente Rede de Petri de Alto-Nível, se mostrou eficiente para os protótipos pretendidos. Conforme a pesquisa foi avançando o nível de complexidade dos modelos se tornou inviável conforme aumentava a complexidade do jogo a ser modelado. Como um dos objetivos dessa tese é fornecer à equipe de *designers* um ambiente para criação de modelo de jogo, sem que seja necessário conhecimento de uma tecnologia tão específica quando Rede de Petri, essa abordagem foi descontinuada e teve início o trabalho com outra tecnologia.

Uma outra abordagem para representar jogos de progressão envolvia a utilização de componentes UML. Após estudos, foi verificado que os personagens de jogo poderiam ser retratados por diagramas de máquina de estado.

Para viabilizar essa abordagem foi construído um protótipo, capaz de ler as ações de um personagem modelado em máquina de estado e gerar seu código, em *Pygame* (PYGAME, 2014), para este personagem. Após um certo avanço essa abordagem também foi desconsiderada, devido a problemas que surgiram relacionados a limitações na representação, que não permitiam a modelagem de um personagem complexo, com muitas ações.

Outros elementos de jogo, como equipamentos, inimigos, itens, fases e regras, seriam ainda mais complexos e precisariam ser caracterizados de outra forma. Como já havia problemas na representação de personagens, certamente também haveria para outros elementos. Deste modo, foi considerado inviável continuar desenvolvendo esse trabalho com máquina de estado.

Diante do observado com as experiências citadas e a complexidade do problema, a solução seguinte foi a considerada mais adequada para atingir os objetivos propostos nessa tese. Iniciou-se, então, o desenvolvimento de um simulador especializado, isto é, voltado para alguns tipos de jogos. Para esse fim, seria necessária a adoção de uma

abordagem procedural, a fim de se obter uma ferramenta com fácil integração e expansão. A linguagem de programação escolhida deveria inevitavelmente ser considerada simples, pois os *designers* de jogos não são necessariamente familiarizados com linguagens de programação.

Por fim, o simulador construído e apresentado nessa tese foi totalmente desenvolvido em Python, uma linguagem de programação lançada no início dos anos 1990 (VENNERS, 2003), com orientação a objetos, procedural, funcional e com uma sintaxe considerada simples.

1.4 Objetivos

O objetivo geral do presente trabalho é propor um ambiente original de simulação de jogos com estrutura de progressão capaz de apoiar o *designer*, ou equipe de *designers*, na construção de um modelo de jogo simulável, visando auxiliar a atividade de balanceamento. São propostos como objetivos específicos:

- Estudo sobre a área de jogos, definição, gêneros, tipos, como avaliar se o jogo consegue entreter, representações de jogos em modelo, balanceamento e possibilidades de simulação;
- Estudo sobre padrões para *design* de jogos;
- Definição da proposta de ambiente de simulação, papel do *designer* e requisitos de uma ferramenta de simulação;
- Implementação de ferramenta de simulação, com base em padrões de projeto de *software*;
- Implementação de componentes auxiliares para facilitar a utilização da ferramenta de simulação; e
- Avaliação da proposta por meio da construção de exemplos de jogos com a ferramenta de simulação.

1.5 Organização

Este trabalho está organizado da seguinte forma. O capítulo 1 faz a introdução do assunto, expõe a hipótese e objetivos da tese.

O capítulo 2 apresenta trabalhos relacionados à área de pesquisa, discute mais profundamente sobre jogos, representações em modelo, balanceamento e possibilidades de simulação.

O capítulo 3 relata as experiências iniciais de pesquisa de tese e outras abordagens realizadas.

O capítulo 4 apresenta a coletânea de padrões para *design* de jogos utilizada para a concepção do ambiente proposto.

O capítulo 5 apresenta a proposta de ambiente de simulação de jogos de progressão.

O capítulo 6 introduz o Pegasus, a ferramenta de simulação desenvolvida para avaliar a proposta.

Os capítulos 7 e 8 apresentam camadas da arquitetura do Pegasus.

O capítulo 9 contém a experiência de uso do simulador, sua validação e modelos de jogos construídos.

Finalmente, o capítulo 10 sumariza esta tese, expõe suas principais contribuições e sugere possibilidades de trabalhos futuros.

2. Revisão da Área de Aplicação

Neste capítulo é apresentada uma revisão dos assuntos relacionados ao tema de trabalho. Inicialmente é feita uma revisão do estudo de jogos, papel do *designer*, gênero de jogos. Na sequência são conceituados as estruturas de jogos e o entendimento sobre jogabilidade. Em seguida é discutida a representação e simulação de jogos. Após, são apresentados os *frameworks System Dynamics* e *Machinations* e, por fim, são destacadas possibilidades de melhoria no poder de expressão.

2.1 Introdução ao estudo de jogos

O estudo dos jogos está comumente associado a palavra ludologia. Esse termo surgiu a partir da combinação da palavra *ludus*, de origem latina, com o sufixo *logia*, que por sua vez tem origem grega. *Ludus* pode ser traduzido como jogar, praticar esporte, passatempo, entretenimento, diversão. E *logia* é um sufixo utilizado para designar um campo de estudo. O termo ludologia vem sendo usado como um termo geral para os estudos e teorias com foco em jogos (FRASCA, 2003).

Quanto à palavra jogo, não existe uma definição única e certeira. Os estudiosos da área possuem diferentes definições para esse conceito. Costikyan, por exemplo, define inicialmente que um jogo não é um *puzzle* porque estes são estáticos. Não são brinquedos, pois estes não possuem objetivos. Não são histórias, pois elas são necessariamente lineares e não requerem a participação do ouvinte. Segundo o autor, um jogo deve obrigatoriamente ter a participação do jogador, ser interativo e possuir objetivos (COSTIKYAN, 1994).

Em um trabalho mais recente, Costikyan lista características que todo jogo deve possuir, para ser considerado, de fato, um jogo. De acordo com o autor, deve existir objetivo; decisões; gestão de recursos; *tokens*; e informação (COSTIKYAN, 2002). Essas restrições fazem com que o autor não considere como jogos atividades que comumente são encaradas como tal, como palavras cruzadas, jogos de azar, simuladores como *SimCity* (ELETRONIC ARTS, 2014) e outras.

Para Schell (SCHELL, 2008) jogos são compreendidos como atividades lúdicas para resolver problemas. Um jogo é visto como uma série de problemas a serem resolvidos e o ato de resolver esses desafios incentivam as pessoas a jogarem. Em outras palavras, jogos possuem características similares a desafios e competições que

proporcionam estímulos positivos em seus jogadores. Essa visão é similar à proposta por (CRAWFORD, 2003), que exige que um jogo seja interativo, tenha objetivos, competição e permita que o jogador possa atacar seu(s) adversário(s) de alguma forma. Deste modo, o autor não considera um jogo atividades como salto à distância ou resolver *puzzle*, pois não há como um jogador influenciar diretamente as ações ou desempenho do adversário.

A noção que jogos são capazes de fornecer sensações prazerosas é fortemente notada nos trabalhos de McGonigal. A autora já palestrou em eventos TED (MCGONIGAL, 2010, 2012a) sobre o assunto e criou o *SuperBetter* (MCGONIGAL, 2012b), uma ferramenta que reduz sintomas de depressão e ansiedade, resultando em bem-estar. Em sua visão, os jogos devem, necessariamente, possuir objetivo; regras; *feedback*; participação voluntária. O objetivo ou meta é o que os jogadores irão conquistar após passarem por todas as etapas do jogo. As regras estabelecem os limites para atingir o objetivo. O *feedback* deve ser um sistema de recompensa a fim de prover sensações prazerosas para que os jogadores mantenham interesse no jogo. A participação voluntária é outro mecanismo para garantir o interesse do jogador em continuar jogando, pois as condições do jogo não devem ser impostas obrigatoriamente (MCGONIGAL, 2011).

Uma definição menos restritiva é dizer que "*games mean gameplay*" (ROLLINGS & MORRIS, 2004). Essa definição se baseia na premissa de que a regra número um de todo *designer* de jogo deve ser "tenha certeza de que o jogo será realmente jogado". Os jogos devem ser capazes de entreter para que sejam jogados sem cair no esquecimento, podendo apresentar fatores surpresa e fornecer sensações agradáveis a seus jogadores (ROLLINGS & MORRIS, 2004).

Salen e Zimmerman montaram um quadro comparativo para identificar elementos que definem um jogo, na visão de cada autor da área (SALEN & ZIMMERMAN, 2004). A tabela a seguir mostra esse quadro.

Tabela 1: Características que um jogo deve possuir, por autor. Adaptado de (SALEN & ZIMMERMAN, 2004)**.**

Elementos de definição de jogo	**Parlett**	**Abt**	**Huizinga**	**Caillois**	**Suits**	**Crawford**	**Costikyan**	**Avedon/ Sutton-Smith**
Prossegue de acordo com regras que limitam jogadores								

Conflito ou competição	■					■		■
Orientado a objetivo ou resultado	■	■			■		■	■
Atividade, processo ou evento		■			■			■
Envolve tomada de decisões		■				■	■	
Não-sério e absorvente			■					
Nunca associado a ganho material			■	■				
Vida comum artificial/segura			■	■		■		
Cria grupos sociais especiais			■					
Voluntário				■	■			■
Incerto				■				
Faz crer / Representacional				■		■		
Ineficiente					■			
Sistema de partes / Recursos e *tokens*						■	■	
Uma forma de arte							■	

A partir dessa comparação, os autores criaram a sua própria: “Um jogo é um sistema no qual jogadores se envolvem em um conflito artificial, definido por regras, que produz um resultado quantificável” (SALEN & ZIMMERMAN, 2004). Essa definição é abrangente o suficiente para englobar outras modalidades como jogos de tabuleiros, cartas, videogames e esportes.

Jesper Juul também possui uma definição abrangente. Para ele, jogos possuem regras fixas, resultado variável, resultado valorizado, consequências negociáveis, ligação do jogador com o resultado e esforço do jogador (JUUL, 2005).

Várias definições foram expostas com o intuito de demonstrar que não existe uma definição mais certa do que outra. Existem diferentes definições, cada qual mais apropriada a seu próprio ambiente ou contexto. A frase “jogos são difíceis de definir,

porém você sabe que é quando vê um" (WIXON, 2006) demonstra que todos, dentro das limitações de seu contexto, são capazes de identificar um jogo ao se depararem com ele.

Este trabalho utiliza a definição abrangente proposta pelo grupo Ludes, Laboratório de ludologia, engenharia e simulação da UFRJ, baseada em definições prévias dos autores mencionados nessa seção: "Jogos são atividades sociais e culturais voluntárias, significativas, fortemente absorventes, não-produtivas, que se utilizam de um mundo abstrato, com efeitos negociados no mundo real, e cujo desenvolvimento e resultado final é incerto, onde um ou mais jogadores, ou equipes de jogadores, modificam interativamente e de forma quantificável o estado de um sistema artificial, possivelmente em busca de objetivos conflitantes, por meio de decisões e ações, algumas com a capacidade de atrapalhar o adversário, sendo todo o processo regulado, orientado e limitado, por regras aceitas, e obtendo, com isso, uma recompensa psicológica, normalmente na forma de diversão, entretenimento, ou sensação de vitória sobre um adversário ou desafio." (XEXÉO *et al.*, 2017).

2.2 *Designer* e processos de *design* de jogo

Durante o processo de elaboração de um novo jogo, o *designer* transita constantemente entre a esfera artística e técnica. Como o jogo é uma espécie de sistema, muitos autores tratam sobre seu processo de construção (BETHKE, 2003) (ROLLINGS & MORRIS, 2004) (CRAWFORD, 1984) (SCHELL, 2008) (SALEN & ZIMMERMAN, 2004) (ADAMS, 2009), alguns para tipos específicos como os jogos sérios (BARBOSA *et al.*, 2014) (BELLOTTI *et al.*, 2012), mas poucos se dedicam a apoiar o processo de *design* com a utilização de métodos formais (GRÜNVOGEL, 2005), embora seja consenso que, devido a não-linearidade da atividade de *design*, não há uma maneira única e eficaz de construção que atenda todos os gêneros e estilos de jogo.

O *designer* ou equipe de *designers* de jogo participam ativamente de todas as etapas do processo de construção. Em seu auxílio podem existir artistas, programadores e outros profissionais envolvidos.

Um dos primeiros *designers* de jogos digitais, Crawford foca na importância de aspectos que circundam o processo. Para ele, é de suma importância iniciar a partir da escolha de um tema e objetivo (CRAWFORD, 1984). Com esses dois elementos bem consolidados, o passo seguinte é imersão no tema através de muita pesquisa. Até esse ponto nada é de fato produzido, somente a partir de então o *design* é iniciado, geralmente

com um esboço de personagem, *level*, mecânica de jogo (CRAWFORD, 1984). As fases seguintes do processo de Crawford incluem escolha da linguagem de programação, estruturas de jogo e tratam de modificações e evoluções a serem realizadas no *design* por possíveis limitações da tecnologia. A implementação, de fato, só inicia após superar essa etapa chamada pré-programação. As demais etapas incluem testes e disponibilização do produto final. A Figura 4 resume esse processo.

Figura 4: Processo de desenvolvimento de Crawford. Adaptado de (CRAWFORD, 1984).

Irish sugere uma sequência de etapas na criação de jogos: concepção, protótipo, pré-produção, produção e controle de qualidade (IRISH, 2005). A metodologia OriGame, resumida na Figura 5, prevê as etapas de concepção, documentação, produção e implementação, com ciclos de testes durante toda a produção para detectar eventuais mudanças (SANTOS *et al.*, 2012).

Figura 5: Metodologia OriGame. Adaptado de (IRISH, 2005)

Salen e Zimmerman (SALEN & ZIMMERMAN, 2004) enfatizam o *design* iterativo, com criação e avaliação de protótipos, para adaptar as decisões baseadas na experiência constante de uso. O modelo apresentado na Figura 6 ilustra a ideia principal do processo conduzido pode diversas iterações.

O processo é iniciado com o conceito do jogo, a partir do qual é construído o primeiro protótipo. O protótipo é então testado e avaliado. O resultado da avaliação decide se é necessário que o protótipo passe por refinamento e uma nova iteração. Após uma série de iterações, o produto final está pronto no momento em que a avaliação do protótipo é considerada satisfatória.

Figura 6: Resumo do processo de *design* de Salen & Zimmerman. Adaptado de

De acordo com Schell (SCHELL, 2008), o primeiro passo no *design* de um jogo é formalizar a ideia inicial, tornando-a algo concreto, definindo os objetivos e as restrições do jogo (Figura 7). Ambos constituem o problema a ser resolvido. A partir daí a equipe de *designers* realiza um *brainstorm* de possíveis soluções e elege uma destas. As fases seguintes incluem a determinação de riscos da solução escolhida, construção e teste do protótipo. O resultado do teste do protótipo, assim como nos processos de *design* apresentados anteriormente, decide se o produto está pronto ou se uma nova iteração é necessária. Nesse caso, um novo problema é explicitado para que sejam geradas novas possíveis soluções.

Figura 7: Ilustração do processo de *design* de Schell. Adaptado de (DZGOEVA, 2016)

O *framework* MDA (em português mecânica, dinâmica e estética, em tradução livre de *mechanics, dynamics and aesthetics*) é uma abordagem formal para tentar aproximar o entendimento de jogos entre o *designer*, desenvolvedor, pesquisador e crítico (HUNICKE *et al.*, 2004).

No campo dos jogos sérios, o desafio para o *designer* é conseguir criar um jogo que leve um ensinamento ao jogador, ao mesmo tempo em que o entretém. Embora diversão não seja um aspecto obrigatório desse tipo de jogo, o engajamento voluntário,

motivado pelo entretenimento, torna a experiência de aprendizado mais proveitosa e duradoura. Existem tentativas de metodologia de desenvolvimento de jogos sérios, como em (BARBOSA *et al.*, 2014), modelos de desenvolvimento para um tipo específico (BELLOTTI *et al.*, 2012) e *framework*, como o DPE (em português *design*, jogo e experiência, em tradução livre de *design, play and experience*) uma expansão do MDA (WINN, 2009).

2.3 Balanceamento de jogo

O processo de balanceamento de jogo é algo custoso, que tem seu início na pré-produção do jogo, mas se estende até a distribuição no mercado. A motivação para se ter um jogo balanceado é evitar o que é conhecido como "jogo quebrado". Quando um jogo está sendo concebido o *designer* naturalmente quer criar uma experiência positiva para os jogadores. Quando o *gameplay* não proporciona essa experiência, pode-se dizer que o jogo está quebrado (FELDER, 2015). Em jogos com cartas, por exemplo, o jogo está quebrado se houver uma carta muito poderosa que concede sempre a vitória ao jogador que a possuir. Em *Counter-Strike*, popular jogo competitivo de tiro em primeira pessoa, a arma AWP é banida por determinados campeonatos por ser a única arma do jogo que mata o alvo com apenas um único tiro, independente do lugar em que ele é atingido (BURGUN, 2011).

Um jogo estar balanceado não significa que todas as opções disponíveis são equivalentes. Se todas as peças de *Xadrez* tivessem o mesmo movimento ele não seria considerado um jogo tão estratégico (FELDER, 2015).

Um jogo desbalanceado pode levar um projeto ao fracasso, pois por mais bonito estética e artisticamente que seja, se suas partes não estiverem em harmonia o jogo corre o risco de não ser jogado (ROLLINGS & MORRIS, 2004). De acordo Rollings e Morris existem três grandes categorias quanto ao balanceamento de jogos. São elas:

- Jogador/Jogador: As partidas entre jogadores devem ser balanceadas de forma que nenhum dos jogadores tenha uma grande vantagem sobre outro. Mesmo existindo o fator sorte esta deve ser aplicada a todos os jogadores.
- Jogador/*Gameplay*: A curva de aprendizado deve estar de acordo com as recompensas que mantém o jogador entretido. O jogador não pode achar que o seu oponente é na verdade sua capacidade de aprender a jogar.

- *Gameplay*/*Gameplay*: Os aspectos do jogo devem ser balanceados entre si. Por exemplo, uma arma que causa um dano muito grande deve ter um custo para compensar seu benefício, como ser mais cara ou deixar o personagem mais lento.

Balanceamento é essencial para que o jogo fique nos extremos de muito fácil ou muito difícil, o que poderia levar a à frustração ou desinteresse (KOSTER, 2005b). A dificuldade de balancear um jogo aumenta de acordo com sua complexidade, no entanto, esse processo é de um grau de dificuldade elevada e consome muito tempo. Dependendo da complexidade do jogo são necessários diversos ciclos *playtests*, que se iniciam somente após as principais partes do jogo estarem prontas.

Dada a importância do balanceamento, é importante trabalhar tal processo de maneiras mais efetivas. Existem estudos que propõem modelos para estudar comportamento do jogador (SONG & KIM, 2012), estudos para tentar automatizar esse processo, como com utilização da inteligência artificial para criar agentes computacionais para jogar o jogo repetidas vezes (JAFFE *et al.*, 2012). Porém, só é possível de se avaliar sobre o jogo já implementado e são soluções *ad-hoc*, isto é, servem somente para o jogo específico para o qual foram projetados.

Em (ANDRADE *et al.*, 2005), os autores aplicam técnicas de aprendizado de máquina em um jogo de luta um contra um, para criar um nível de dificuldade dinâmico, que faz com que o oponente se torne mais fácil ou difícil de acordo com o comportamento do jogador.

Uma técnica bastante utilizada pela indústria é criar um seletor de dificuldade no início do jogo. Quanto mais opções disponíveis significa mais tempo que foi empregado durante a produção do jogo. Em termos de balancear um jogo, a inteligência coletiva dos jogadores é melhor que um excelente *designer*, por melhor que ele seja (BURGUN, 2011).

Dado esse cenário, a realização de simulações, com indicadores como resultados, pode economizar tempo de produção do jogo e auxiliar o processo de balanceamento que é tão importante para o jogo.

2.4 Gêneros de jogos

No estudo de jogos, falar sobre gêneros é como uma tentativa de classificação que dificilmente haverá um consenso. Muitos autores já discutiram e concluíram que,

dependendo do observador, um jogo pode ser classificado sob um gênero ou sob outro. Ou ainda, dependendo da classificação os gêneros e subgêneros existentes podem variar.

Järvinem dedica um capítulo inteiro de sua tese (JÄRVINEN, 2009) para discutir sobre gêneros de jogos. Sob o aspecto independente de tecnologia, o autor classifica jogos como:

- Jogos de cartas e dados
- Jogos de caneta e papel
- Jogos de tabuleiro
- Jogos de computador ou console (jogos digitais)
- Jogos portáteis (jogos digitais para celular ou console portátil)

A classificação acima foi incluída com o intuito de relembrar que jogos são muito mais do que somente os videogames (jogos digitais), que são o estudo principal dessa tese.

Quanto aos videogames, o *site Gamespot* (www.gamespot.com) possuía em 2006 os seguintes gêneros: Ação, Aventura, Corrida, *Puzzle*, *Role-playing game* (RPG), Simulação, Esporte e Estratégia (JÄRVINEN, 2009). Uma visita mais recente (em 02/01/2018) ao site revelou 68 classificações como, por exemplo, as novas Pinball, Música/Ritmo, Sobrevivência, Surfe e outras. São 60 novas classificações no período de aproximadamente uma década. É notável que o *Gamespot* não se preocupou em dividir essa classificação em uma hierarquia, já que existem as classificações Esportes, Futebol, Surfe, Boxe etc.

Quanto a jogos de tabuleiro, o *site Boardgamegeek* (www.boardgamegeek.com), popular comunidade de jogos desse tipo, utiliza a denominação categorias ao invés de gêneros, mas na realidade é uma mistura de temas, mecânicas, tecnologias e outras informações. Atualmente são 84 categorias. A lista completa das classificações do *Boardgamegeek* e do *Gamespot* estão presentes no Anexo I.

De acordo com (JÄRVINEN, 2009), o processo de criar classificações de gêneros e subgêneros é dependente do aspecto que se quer abordar. Deste modo, as classificações de gêneros em relação ao aspecto dos componentes de jogo são diferentes das classificações de gêneros para o aspecto do conjunto de regras, da mecânica, ou do tema.

Na década de 1980, quando os jogos digitais eram ainda rudimentares, definir um gênero e classificar os jogos sobre este gênero configurava-se como tarefa mais simples, tanto pela pouca quantidade de jogos quanto pela tecnologia da época. Em 1984, por

exemplo, Crawford propôs uma taxonomia para dividir os jogos digitais sobre duas grandes categorias, com subcategorias: Habilidade-e-Ação (*Skill-and-Action)* e Jogos de Estratégia. A primeira possui como subcategorias: Combate, Labirinto, Esportes, *Paddle*, Corrida, Miscelânea. E as subcategorias dos Jogos de Estratégia são: Aventura, D&D (*Dungeons and Dragons*), Guerra, Jogos de Sorte (*Games of Chance*), Educacionais, Interpessoais (CRAWFORD, 1984) .

O estudo de gêneros de jogos digitais precisa ser encarado com mente aberta e versatilidade, pois trata-se de um meio novo, interativo e que continua em expansão e evolução. Pode-se dizer que sempre haverá algo novo para aprender sobre a estética, economia, impacto social e cultural desse novo meio (CLEARWATER, 2011).

2.5 Emergência e Progressão em Jogos

De acordo com Juul, os jogos digitais podem ser encaixados em duas estruturas básicas de jogos: emergência e progressão (JUUL, 2002). O objetivo dessa seção é definir de maneira clara esses dois conceitos para o leitor.

O termo emergência remete a ideia de que um jogo é especificado como um pequeno conjunto de regras que combinam e produzem grandes quantidades de variações de jogo, que os jogadores então criam estratégias para lidar (JUUL, 2002). Devido a suas características, os jogos de emergência conseguem ser jogados por várias vezes, dado que as partidas serão sempre diferentes. A característica de emergência pode ser notada em jogos como xadrez, *poker*, jogos de cartas, jogos de estratégia, jogos de ação, esportes, torneios.

De maneira diferente, progressão trata de uma sequência bem controlada de eventos. Basicamente, o *designer* define que desafios um jogador encontra em cada etapa ou *level*. O jogador precisa superar tais eventos, em uma dada sequência, para poder avançar no jogo (JUUL, 2002). A superação de eventos geralmente envolve o cumprimento de algum objetivo, que pode ser derrotar todos os inimigos de uma área, atingir uma localidade, encontrar algum objeto etc. É bem raro encontrar jogos de tabuleiro que seguem esse formato, sendo mais apropriado para jogos que contam uma história. Podem ser citados com jogos com progressão *Super Mario, Resident Evil*, *Phantasy Star*.

Os jogos de emergência costumam possuir guias de estratégias, enquanto que jogos de progressão costumam ter *walkthroughs*, isto é, um passo-a-passo das ações que precisam ser realizadas para completar o jogo.

Muitos jogos modernos se encaixam nas duas definições contendo, portanto, elementos de emergência e progressão (DORMANS, 2012b) (JUUL, 2002). Podem ser citados como exemplos as séries de jogos *GTA, Assassin's Creed, Red Dead Redemption, The Division, Destiny*. São jogos com mundo aberto, em que o jogador controla um personagem com liberdade para explorá-lo e realizar diversas atividades. No entanto, existe uma sequência de missões que, ao serem executadas, vão contando a história do jogo e desbloqueando novos locais. Essas missões são geralmente chamadas de missões principais e as outras de secundárias. Cada "missão" possui sua complexidade e seus próprios elementos de emergência e suas estratégias e, ainda assim, o jogo contém progressão e conta sua história.

2.6 *Gameplay* e *Playability*

As palavras *Gameplay* e *Playability*, em português, provavelmente seriam traduzidas para "jogabilidade". Para evitar ambiguidade entre os termos essa tese só os utiliza em inglês.

O termo *gameplay*, mesmo em inglês, pode ter o seu significado controverso, dado que alguns autores ainda confundem com o significado de *playability*. Essa tese utiliza para *gameplay* a definição "é a interação formalizada que ocorre quando os jogadores seguem as regras de um jogo e experimentam seu sistema através do jogar" (SALEN & ZIMMERMAN, 2004). De maneira resumida, considera-se por *gameplay* a interação entre jogador(es) e jogo.

A definição de *playability* adotada nesse trabalho é "um conjunto de propriedades que descrevem a experiência do jogador em um jogo, cujo principal objetivo é proporcionar prazer e entretenimento, independente se o jogador joga sozinho ou em companhia" (SÁNCHEZ *et al.*, 2009). *Playability* se refere a todas as experiências que um jogador pode sentir ao interagir com um sistema de jogo. Dessa forma, pode-se dizer que *playability* é utilizado para tentar medir a qualidade do *gameplay* de um jogador.

Sweetser e Wyeth criaram um modelo para avaliar o grau de divertimento do jogador no jogo (SWEETSER & WYETH, 2005). Em seu modelo os critérios avaliados são:

- Concentração: jogos devem demandar concentração e o jogador deve ser capaz de se concentrar no jogo;
- Desafio: jogos devem ser suficientemente desafiadores e condizer com o nível de habilidade do jogador;
- Habilidade do jogador: jogos devem suportar o desenvolvimento da habilidade do jogador, desde o novato até o *expert*;
- Controle: jogadores devem sentir uma sensação de controle sobre suas ações no jogo;
- Objetivos claros: jogos devem fornecer ao jogador objetivos claros nos momentos apropriados;
- Feedback: jogadores devem receber *feedback* nos momentos apropriados;
- Imersão: jogadores devem esquecer dos seus problemas e se sentir imersos no jogo, tendo a noção de tempo alterada;
- Socialização: Jogos devem criar oportunidades para interação social entre os jogadores.

Mais recente, Sánchez propôs um conjunto de sete atributos, cada um composto por algumas propriedades, para caracterizar *playability*¸ conforme mostra a Figura 8. Esses atributos são: Eficácia; Aprendizado; Imersão; Satisfação; Motivação; Emoção; Socialização (SÁNCHEZ *et al.*, 2012). Mais detalhes sobre esses atributos são fornecidos em capítulo posterior.

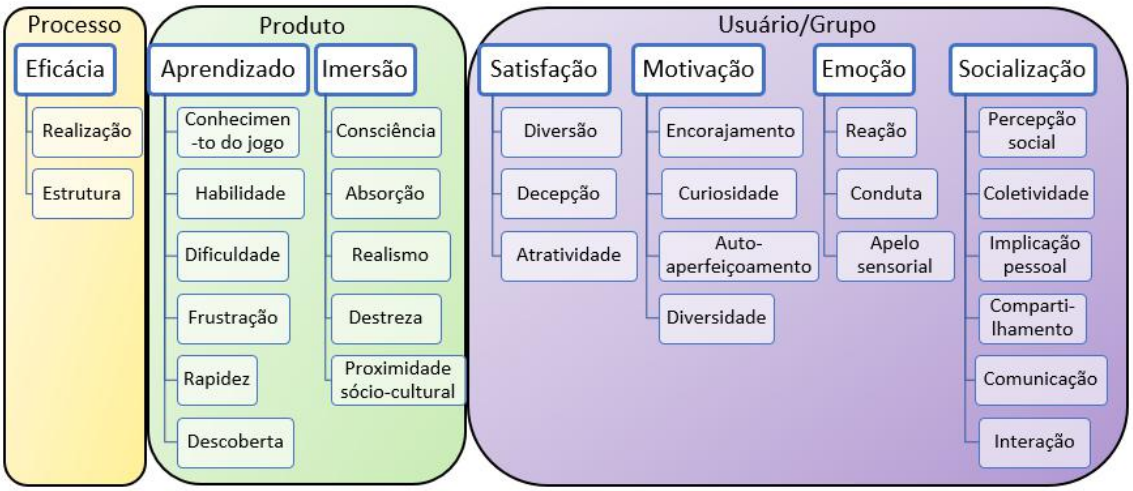

Figura 8: Atributos e propriedades do modelo de *playability*. Adaptado de (SÁNCHEZ *et al.*, 2012)

2.7 Representação de jogos

Alguns jogos exercitam a capacidade de explorar cenários, tomar decisões, a precisar a mira num alvo, resolver *puzzles*, tornando-os bastante complexos. Um *designer* de jogo consegue pensar e descrever jogos simples sem dificuldade, como o jogo da velha. É possível, inclusive, representá-lo em uma linguagem de descrição de jogo como GDL. Jogos mais complexos, no entanto, são difíceis de serem representados.

Game Description Language (GDL) foi concebida pela *Stanford University* como parte do projeto *General Game Playing* (GGP). Este projeto tinha como objetivo a criação de uma plataforma para que programas de inteligência artificial fossem capazes de jogar mais de um tipo de jogo (GENESERETH *et al.*, 2005). Antes dessa concepção, era comum a criação de programas altamente especializados, com algoritmos avançados, para jogar somente um jogo. Provavelmente *Deep Blue*, um supercomputador e *software* da década de 1990 criados pela IBM para competições de xadrez, seja o computador mais famoso desse segmento. O *software* do *Deep Blue* é tão especifico que ele não consegue, por exemplo, jogar uma partida de damas.

GDL é uma linguagem formal, declarativa, para descrever alguns tipos de jogos determinísticos com informações bem definidas. Em GDL, jogos são modelados como máquinas de estado, em que cada estado do jogo é descrito como um conjunto de fatos. As regras do jogo são descritas por regras lógicas que definem os estados sucessores, com base no estado corrente e nos movimentos dos jogadores (KAISER, 2005).

Variações da GDL foram criadas, de maneira que a primeira também passou a ficar conhecida como Stanford GDL. De acordo com Cameron Browne, criador da *Ludi GDL* (BROWNE, 2008) (BROWNE, 2011), a GDL mais utilizada é provavelmente a *Zillions Rules File* (ZRF), formato utilizado pela aplicação comercial *Zillions of Game* (MALLETT & LEFLER, 1998). Os jogos que podem ser descritos com GDL possuem algumas propriedades. São elas:

- O jogo é baseado em turnos;
- O jogo é finito. Os movimentos de um jogador no turno são finitos, e o jogo sempre irá terminar em um número finito de turnos;
- O jogo tem informação perfeita. Em cada turno, todos os jogadores sabem tudo o que há para saber sobre o estado do jogo;
- O jogo é determinístico, sem elementos aleatórios.

Os dois últimos requisitos não existem na extensão GDL-II, criada para abranger jogos em que jogadores possuam informações assimétricas como, por exemplo, jogos de carta, em que a mão do jogador não é conhecida, ou jogos que envolvem elementos de sorte, como jogar dados ou girar roleta. GDL-II adiciona dois novos elementos à linguagem: aleatoriedade, para modelar jogos que possuem elementos de sorte; e percepção, para especificar regras de que informações os jogadores possuem em quais condições (THIELSCHER, 2010) (THIELSCHER, 2011).

A GDL é uma linguagem poderosa, porém centralizada na mecânica do jogo, nas suas regras e funcionamento, pois foi projetada para atender a aplicações de inteligência artificial.

Quanto à representação visual, os jogos descritos em GDL podem ser representados não somente num diagrama de máquina de estado, mas também por meio de uma *Propositional Net* (GENESERETH & THIELSCHER, 2014). Essa estrutura, também chamada de *PropNet*, foi apresentada pela *Stanford University* como uma maneira alternativa de expressar as regras de um jogo.

Raramente os estados são compreendidos como entidades monolíticas. Em geral, os estados são caracterizados em termos de proposições que são verdadeiras nesses estados. Conforme as ações são executadas, algumas proposições se tornam verdadeiras e outras se tornam falsas. Em uma *PropNet* as proposições e ações são os nós em vez de estados e, os nós são intercalados com conectores lógicos e transições (GENESERETH & THIELSCHER, 2014).

Para fins de comparação entre as representações descritas, a Figura 9 apresenta na parte de cima um exemplo de diagrama de máquina de estado e, na parte de baixo, um exemplo de *PropNet*. Um dos benefícios desse tipo de representação é a compacidade, pois um conjunto de *n* proposições, se levado em conta todas as diferentes combinações de valores lógicos, corresponde a um conjunto de 2^n estados.

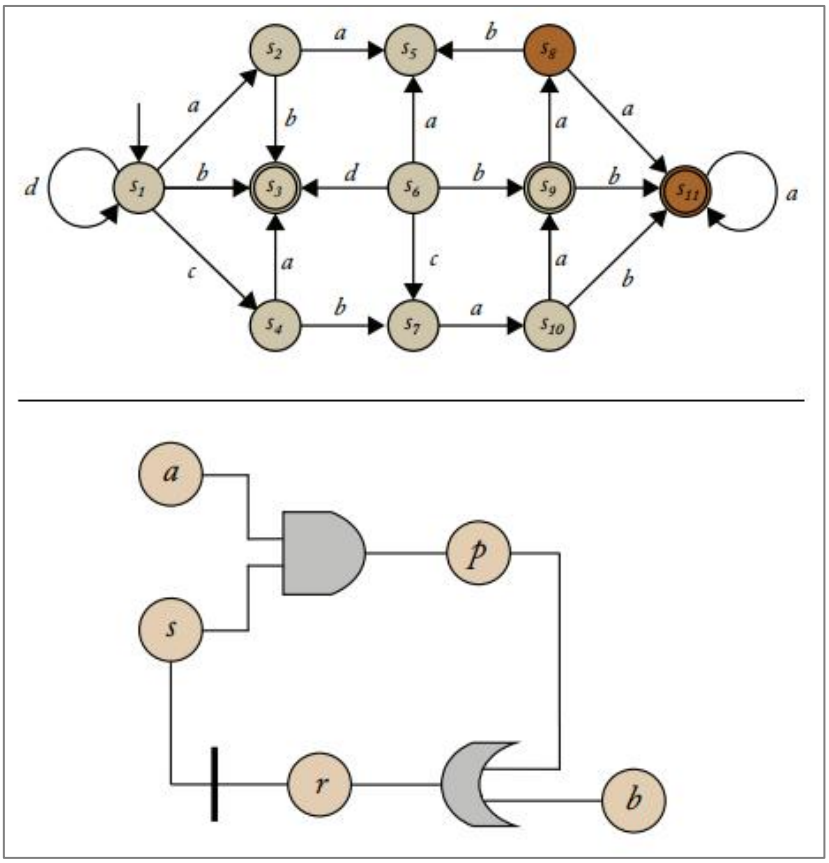

Figura 9: Máquina de estado e *PropNet*. Retirado de (GENESERETH & THIELSCHER, 2014)**.**

Stéphane Bura propõe uma gramática para que os *designers* de jogos descrevam jogos de tal forma que o modelo possa expressar os tipos de interações entre os jogadores, seja capaz de representar como diferentes partes do jogo são ligadas, e seja simples o suficiente para ser "rabiscado em um guardanapo" (BURA, 2006).

A gramática proposta por Bura tem influência de suas experiências com cibernética, possui elementos como transições, *tokens*, sumidouros, estados e outros. Como o próprio autor afirma, sua gramática é apenas um primeiro passo para que outros possam se beneficiar. Alguns dos elementos propostos pela gramática de Bura estão presentes no *Machinations*.

O *Machinations* é um *framework* criado por Joris Dormans, com assistência do próprio Bura. O *Machinations* constitui uma maneira de representar jogos emergentes evidenciando sua economia interna, com possibilidade de simulação. Ele funciona como uma linguagem visual composta por seu próprio diagrama e seus componentes. Através destes, o *designer* tem a possibilidade de criar a representação visual da economia interna de seu jogo (DORMANS, 2012b).

A mecânica dos jogos e suas características estruturais não são imediatamente visíveis na maioria dos jogos. Apesar de algumas mecânicas serem aparentes para os jogadores, outras estão ocultas no código do jogo. *Machinations* se propõe a representar a mecânica de jogos de uma maneira acessível, mantendo as características estruturais e o comportamento dinâmico dos jogos. A Figura 10 resume o *Machinations* e seus principais componentes.

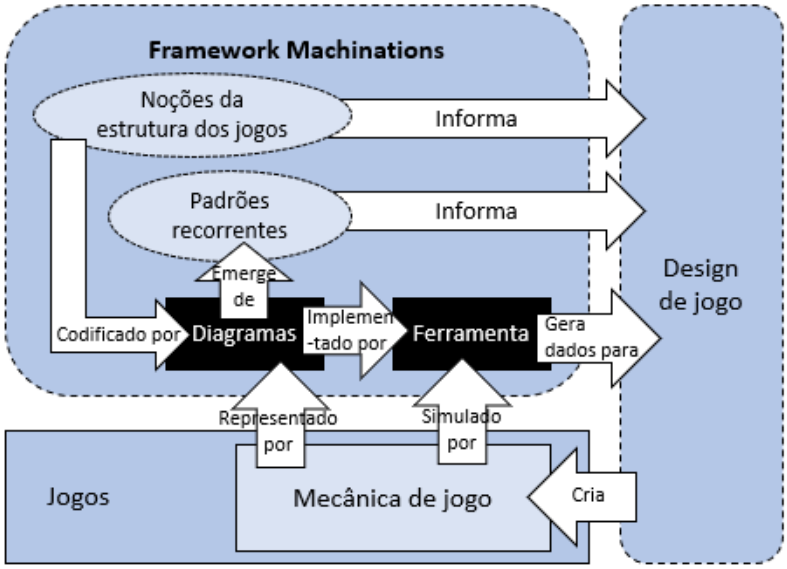

Figura 10: O *framework Machinations*. Adaptado de (ADAMS & DORMANS, 2012)

Dormans defende o *design* de jogos orientado a modelos, atestando que confere flexibilidade e fluidez ao processo. Apesar de alguns *designers* serem relutantes com essa forma de trabalho, o autor alega que modelos poderiam ser utilizados para desenvolver ferramentas automatizadas, possibilitando o aumento da eficiência de desenvolvimento (DORMANS, 2012a).

Devido a sua característica de simulação de modelo, o *Machinations* é abordado novamente na seção 2.8.2.

2.8 Simulações de jogos

Raph Koster observa que existem jogos que seguem a mesma mecânica. Ao jogar um nível repetidamente, o cérebro humano começa a entender tal padrão e tenta aperfeiçoar cada vez mais, para que o jogador supere o nível da maneira mais eficaz possível. Dessa forma, uma vez que o jogador consegue entender e dominar tal técnica, o jogo pode se tornar entediante (KOSTER, 2005b).

Jogos que são extremamente fáceis ou extremamente difíceis não conseguem prender a atenção de jogadores por muito tempo. Conforme exposto, para atingir esse objetivo o jogo deve ser capaz de entreter. Os seres humanos possuem interesses distintos em todos os aspectos e dimensões da sociedade e, consequentemente, o gosto por tipos de jogos também varia de jogador para jogador. No entanto, há um consenso geral de que jogos bem balanceados conferem melhor *playability* ao jogador e conseguem prender sua atenção. As características necessárias para que um jogo esteja balanceado vão ao encontro das propriedades de *playability,* destacadas na seção 2.6.

Simular o *gameplay* é uma maneira de avaliar a dinâmica de um jogo, o seu desenrolar, prever possíveis situações. Esta abordagem é um artifício eficaz que pode ser empregada quando o jogo ainda está em fase de desenvolvimento para descobrir se ele está bem balanceado (SALGE & MAHLMANN, 2010).

Existem abordagens de *design* de jogos que defendem a realização de simulações quando o jogo ainda se encontra nas fases iniciais de desenvolvimento. Por meio destas, é possível detectar falhas e oportunidades de melhorias em suas dinâmicas e recursos (NUMMENMAA *et al.*, 2009). No processo de criação de jogo, assim como no processo de criação de qualquer sistema computacional, o quanto antes os problemas forem identificados melhor será, uma vez que evita retrabalho.

Simulação é uma técnica frequentemente empregada em diversas áreas de aplicação como, por exemplo, para analisar processos de produção em série, otimizar redes de transportes, avaliar comportamentos de usuários quando submetidos a diferentes situações, e outros usos (SOKOLOWSKI & BANKS, 2008).

O emprego de simulação em conjunto com jogos também pode ser encontrado em casos da utilização de um jogo para simular situações reais e, com isso, fornecer algum tipo de aprendizado aos envolvidos no processo (RUBEN, 1999) (VOGEL *et al.*, 2006).

As simulações de jogos costumam ser bem especializadas, restritas a seu domínio. Um exemplo dessa especificidade é o trabalho de Syriani e Vangheluwe. Os autores consideram que a movimentação dos personagens no jogo pode ser representada por meio de eventos discretos e, partindo desse princípio, propuseram uma abordagem para realizar simulações visando melhorar a *playability* (SYRIANI & VANGHELUWE, 2008). O estudo de caso foi feito sobre o jogo *Pac-Man* (NAMCO, 1983), que possui mecânicas simples de movimentação, para simular um modelo onde as mudanças de estado no jogo são descritas como transformações baseadas em regras para um grafo de estado. O

objetivo era variar a maneira como o fantasma, inimigo do *Pac-Man*, se move para atingir a sua configuração ótima de velocidade de acordo com a velocidade de reação de cada tipo de usuário (lento, normal, rápido e muito rápido). Apesar de ser um trabalho com simulações, o controle do personagem principal precisou ser realizado por humanos.

Em (NUMMENMAA *et al.*, 2009), os autores propõem o *design* de jogo apoiado por ferramenta de simulação. Os autores utilizam um pacote de *software* de simulação chamado *DisCo*, e descrevem um estudo de caso em que um jogo simples real é "transcrito" na linguagem *DisCo*. Trata-se de uma abordagem bem objetiva, que deixa de lado todos os detalhes desnecessários do jogo para produzir um modelo altamente simplificado do sistema de jogo. Ao final do processo, o *designer* pode utilizar o resultado da simulação como base para visualizar alguns comportamentos do sistema num nível mais alto do que uma prototipagem comum.

Outra abordagem para simulação de jogos, ainda que apenas na concepção teórica, foi proposta por Grünvogel, após notar que a dificuldade de simulação de um jogo aumenta conforme sua complexidade, propondo assim a criação de modelos formais simples que atuem sobre aspectos selecionados de um jogo (GRÜNVOGEL, 2005). O autor introduz um formalismo para *design* de jogos em que seriam construídos pequenos submodelos. Gradativamente, estes modelos iriam sendo combinados em sistemas mais complexos e, seu conjunto, representaria o jogo como um todo.

As subseções seguintes apresentam dois *frameworks* que podem ser utilizados para simulações de alguns tipos de jogos: *System Dynamics* e *Machinations*.

2.8.1 *System Dynamics* como ferramenta de simulação

System Dynamics é uma abordagem para entender o comportamento não-linear de sistemas complexos de acordo com o tempo, utilizada para modelar, entender e discutir questões e problemas complexos. Criada na década de 1950 pelo professor Jay W. Forrester, do Massachusetts Institute of Technology (SYSTEM DYNAMICS SOCIETY, 2015), *System Dynamics* apoia a identificação de fatores que influenciam o comportamento de um sistema ao longo do tempo.

A abstração de pensar em estoques e fluxos dentro de processos e as relações entre eles permite encaixar modelos de *System Dynamics* em diversas áreas da sociedade (WOLSTENHOLME, 2005). O trabalho (HIRSCH *et al.*, 2005) analisa as reformas da saúde nos Estados Unidos que ocorreram com o passar dos anos, e identifica os ganhos

proporcionados com elas. A Figura 11 ilustra um modelo *System Dynamics* representando o primeiro estágio da saúde da população americana. Os modelos são incrementados e utilizados como ferramentas de apoio para entender e identificar novas possibilidades de melhoria no sistema de saúde.

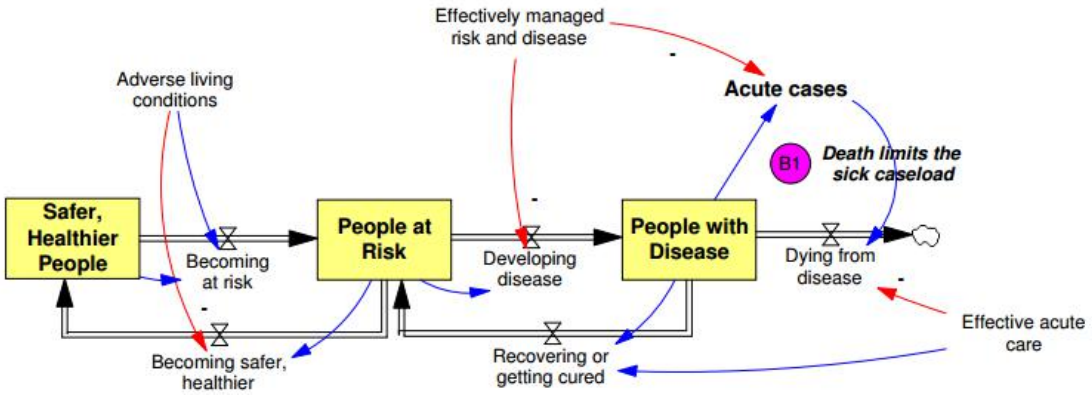

Figura 11: Dinâmica de saúde da população. Retirado de (HIRSCH *et al.*, 2005)**.**

O *Beergame* é, provavelmente, o mais famoso jogo baseado em *System Dynamics* (RIEMER, 2012). Foi criado na década de 1960 pelo próprio Jay W. Forrester para pesquisar o efeito das estruturas de sistemas no comportamento das pessoas, sendo hoje utilizado por estudantes com o intuito de experenciar problemas típicos de um sistema de abastecimento em cadeia (RIEMER, 2008) (DIZIKES, 2013). O *Beergame*, apesar de várias adaptações ao longo dos anos, tem como objetivo simular quatro estágios de um sistema de abastecimento de cadeia: produção da bebida, seu deslocamento por dois níveis de fornecedores, até a entrega ao vendedor final. Cada jogador atua em um desses estágios e deve realizar pedidos ao estágio anterior. Como os jogadores não podem se comunicar verbalmente para informar com antecedência a quantidade de bebida que irão solicitar/fornecer, com o passar do tempo ocorre, inevitavelmente, o efeito chicote entre o que é pedido e o que é fornecido entre um estágio e o seguinte.

Muitas ferramentas de simulação baseadas em *System Dynamics* são utilizadas como forma de aprendizado de sistema, a fim de se obter uma maior compreensão e possibilidade de melhoria de performance (FORD, 1998). Uma fase importante nesse processo é o *debriefing*, momento após o término da execução de uma atividade em que se analisa e discute o processo, buscando maior compreensão.

A Figura 12 retrata o ciclo de *feedback* influenciando as decisões que tomamos no mundo real, que por sua vez geram novos *feedbacks*. A figura mostra que as decisões também são influenciadas por estratégias e regras de decisões, sendo estas construídas a

partir de modelos mentais do mundo real. As ferramentas de *System Dynamics* podem ser empregadas nos processos de simulação e aprendizado, contextualizados na área destacada.

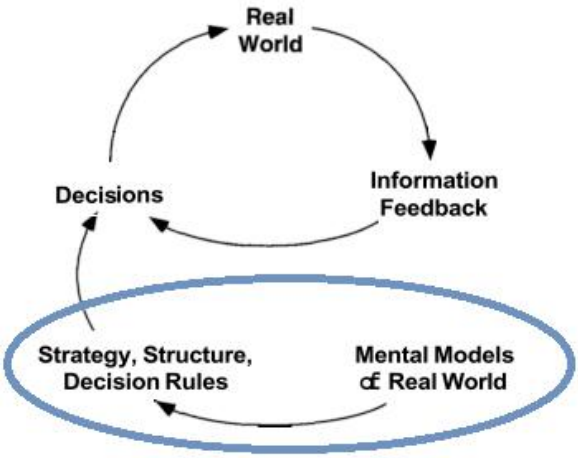

Figura 12: Ciclo de aprendizado. Adaptado de (STERMAN, 2000)**.**

Por ser uma abordagem tão generalizada, é possível utilizar *System Dynamics* para realizar macro simulações de jogos, isto é, em um alto nível de abstração. Dessa forma, a representação fica ausente de detalhes específicos que seriam necessários para uma representação mais fiel do modelo.

Conforme relatado em (VAN DAALEN *et al.*, 2014), quando se procura a relação de jogos e *System Dynamics*, a maioria da literatura encontrada se refere a aprendizados abordando jogos de simulação, como *Microsoft Flight Simulator* (MICROSOFT STUDIOS, 2011), por exemplo. Alguns trabalhos publicados falam sobre taxonomias ou definições, alguns outros sobre *game design* e algum tipo de simulação, em alto nível. Segundo os autores, apesar de ser considerada relativamente fácil a concepção de um jogo baseado em *System Dynamics*, o desenvolvimento de um jogo realmente eficaz não é trivial e são poucas as histórias de sucesso.

2.8.2 *Machinations* como ferramenta de simulação

Jogos de aposta ou de jogos azar, tais como roleta ou *blackjack* possuem mecânicas simples e probabilidades bem definidas. Para jogos desses tipos é possível calcular as chances de vitória dos jogadores, mesmo sem jogar as partidas. Para jogos mais complexos, incluindo jogos com muitas tomadas de decisão ou fatores aleatórios, esse cálculo já se torna mais difícil de ser realizado. Nesses casos, a maneira mais eficaz

de avaliar o balanceamento do jogo é jogando várias vezes e, aos poucos, ir realizando ajustes até que se obtenha o nível de balanceamento considerado justo para todos os jogadores, dado o contexto do jogo.

Conforme apresentado em 2.8 o balanceamento do jogo é um fator importante para que ele prenda a atenção de seus jogadores. O *framework Machinations* permite a criação de modelos que representam jogos, configuração de alguns elementos e execução de múltiplas simulações de um jogo, com o uso de jogadores artificiais. Ao final, é informada a quantidade de vitórias de cada jogador.

A figura a seguir ilustra o diagrama de um modelo do jogo *Monopoly* (HASBRO, 1935) para dois jogadores. É possível notar que os autores precisaram assumir alguns valores para tornar a simulação possível. Foi assumido que a cada propriedade adquirida por um jogador acrescenta 4% de chance do outro jogador pagar dinheiro ao primeiro a cada rodada (ADAMS & DORMANS, 2012). Ainda, os autores também definiram que a cada rodada cada jogador possui 10% de chances de adquirir uma nova propriedade, caso tenha dinheiro disponível.

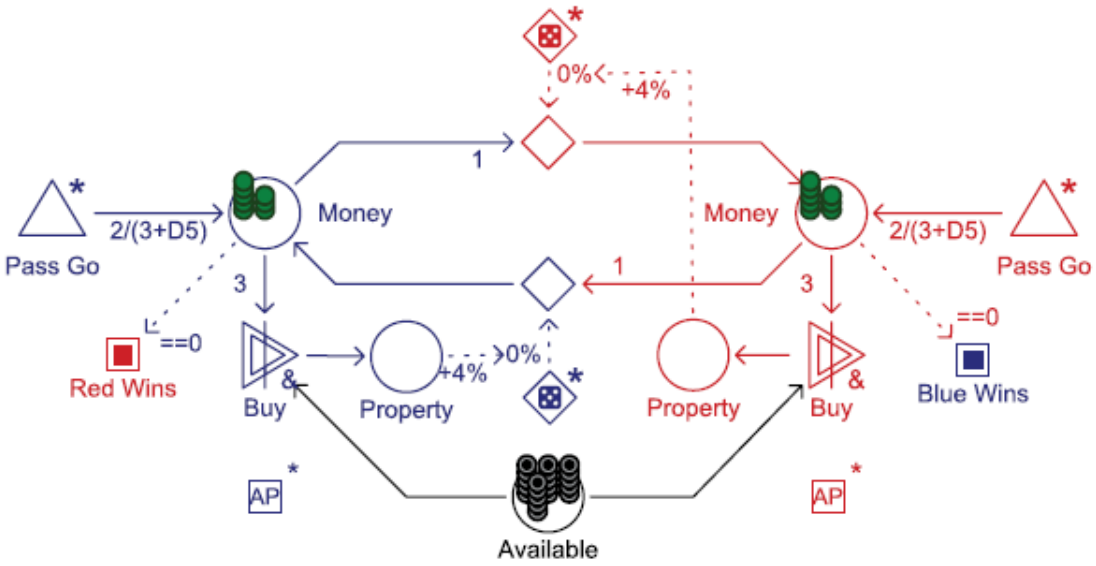

Figura 13: Simulação para Monopoly com dois jogadores. Retirado de (ADAMS & DORMANS, 2012)

O *Machinations* não se limita a trabalhar com jogos em níveis abstratos. É até possível, mas seu potencial se faz realmente perceptível pela capacidade de representação e simulação de níveis mais baixos, como a mecânica e a economia interna dos jogos. Essa característica faz do *Machinations* uma ferramenta com características voltadas para

jogos de emergência, mas com dificuldade para representações de jogos com características de progressão.

A ferramenta permite a utilização de *tokens* em cores distintas, para que possam representar recursos como vidas, pontos, moedas, itens e outros. Contudo, tais representações são sempre simples, unitárias.

Devido a sua notação gráfica para expressar regras da economia de jogos, o *Machinations* despertou grande interesse da comunidade científica da área. Uma linguagem específica de domínio (DSL) denominada *Micro-Machinations* foi criada para detalhar e formalizar o subconjunto de recursos significativos do *Machinations*, além de acrescentar novos recursos como modularização (KLINT & ROZEN, 2013). O próprio Dormans contribuiu com o *Micro-Machinations* e, posteriormente, descreveu com Rozen como utilizar a modularização para o reuso de jogos construídos no *Machinations* (ROZEN & DORMANS, 2014).

Uma das desvantagens do *Machinations* é ser construído totalmente em Flash (ADOBE, 1996), uma tecnologia em declínio. Pode-se citar o *Rachinations* como variação ao *Machinations*, batizado com esse nome para fazer uma menção à combinação da linguagem de programação *Ruby* com o *Machinations* (ALMEIDA, 2015a). O *Rachinations* possui dois componentes principais: uma DSL baseada em *Ruby* para descrever jogos de maneira similar a um diagrama construído no *Machinations*; e uma *engine*, responsável por executar a simulação do diagrama. Não há interface gráfica, de forma que todo o diagrama deve ser descrito por meio da DSL. O projeto possui código aberto e pode ser acessado em (ALMEIDA, 2015b).

2.8.3 Possibilidades de melhoria do poder de expressão

Jogos podem ser tão complexos que é difícil imaginar a possibilidade da existência de uma ferramenta que permita realizar simulações com alto nível de detalhamento. Uma ferramenta como *Machinations*, voltada à criação de modelo com representação da estrutura interna do jogo, revela uma maneira detalhada de se analisar o que se passa no jogo, pois é possível representar os recursos disponíveis de cada jogador a cada passo e, partir desse cenário, analisar o que pode ser melhorado a fim de se obter o melhor balanceamento.

Existem, no entanto, algumas limitações por parte do *Machinations*. O uso de *tokens* como representação dos recursos provê abrangência na representação, porém estes

são simples e não conseguem retratar certos recursos com expressividade. Seria interessante permitir a criação especializada dos recursos de jogo. Dessa forma, estes poderiam conter especificidade, comportamento próprio, recursos derivados e dependentes de outros recursos. Essa proposta de melhoria certamente aumentaria o poder de representação da economia interna do jogo, fazendo com que as simulações estivessem mais próximas da realidade e, consequentemente, tornando os resultados mais confiáveis.

Outro problema identificado é a complexidade para a construção de modelos de representação do jogo. Quanto mais complexo é um jogo, maior e mais intrincado é o diagrama que o representa, de forma que sua criação não é uma atividade trivial. Dependendo da complexidade do jogo pode até mesmo ser inviável a construção do diagrama que o representa.

Nesse ponto, uma possibilidade de melhoria seria a identificação de construções recorrentes em jogos e, com base nos padrões de jogos, elaborar novos componentes baseados nestes. Tais componentes poderiam substituir conjuntos de elementos e facilitar a criação de diagramas por parte do *designer*.

3. Experiências Iniciais

Durante as pesquisas para a Tese houveram experiências iniciais com outras tecnologias, relatadas nesse capítulo. Uma das experiências foi a tentativa de criação de uma ferramenta baseada em Redes de Petri. A outra experiência envolvia a geração automática parcial de código de jogo a partir de modelos de máquinas de estado.

3.1 Experiência com Redes de Petri

Redes de Petri compõem um poderoso formalismo de modelagem em ciência da computação, engenharia de sistemas e outras disciplinas, combinando teoria matemática com representação gráfica do comportamento dinâmico de sistemas (PETERSON, 1981) (WANG, 2007). O Anexo II contém mais detalhes sobre essa tecnologia, os tipos e variações de redes existentes, sua linguagem e ferramentas.

Com o intuito de verificar a viabilidade da construção e execução do modelo de um jogo simples em Rede de Petri, foi construído na ferramenta *CPN Tools* o modelo de um jogo em que os possíveis cenários já são conhecidos. O jogo escolhido foi o *Jokenpo*, também conhecido no Brasil por Pedra-Papel-Tesoura. Este jogo é comumente utilizado para decisões rápidas entre duas pessoas, da mesma forma o par-ou-ímpar.

Em *Jokenpo* os jogadores devem escolher um dos três elementos e "jogar" simultaneamente com suas mãos. Os possíveis elementos são pedra (punho cerrado), papel (mão espalmada) e tesoura (indicador e médio formando um V). O jogo tem apenas três possíveis resultados: vitória do jogador 1; vitória do jogador 2; empate. Cada elemento ganha de um e perde de outro, de forma que não existe uma escolha que dê ao jogador mais chances de vencer. O empate ocorre quando ambos os jogadores escolhem o mesmo elemento (WORLD RPS SOCIETY, 2015).

Dentre as três combinações em que ocorre vitória de um dos jogadores, costuma-se dizer que "pedra quebra a tesoura", "tesoura corta o papel" e "papel cobre a pedra". A figura a seguir apresenta tais combinações.

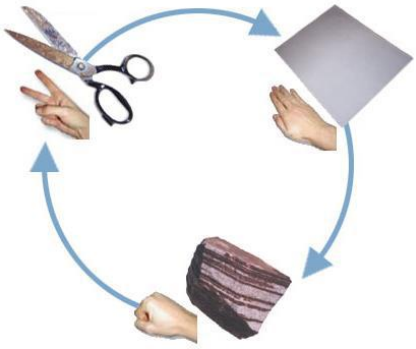

Figura 14: Combinações no *Jokenpo*. Retirado de (LIVESCIENCE, 2011)

Em *Jokenpo* o único recurso existente é a jogada a ser escolhida pelo jogador. Para simbolizar isso foi criada a cor HAND, cujas únicas possibilidades são R (rock), P (paper) ou S (scissors). O resultado do jogo é contabilizado pela cor RESULT. A função *winner* contém a lógica que define o resultado do jogo. A seguir são apresentados os *colorsets* e o modelo que foram construídos na ferramenta *CPN Tools* para representar o jogo.

```
colset HAND = with R|P|S;
var h1, h2: HAND;
colset RESULT = with P1|P2|E;
var r:RESULT;
fun winner(h1:HAND,h2:HAND) = (
let
  val j = (h1,h2);
in
  if(j=(R,S) orelse j=(P,R) orelse j=(S,P)) then P1
  else if(j=(R,P) orelse j=(P,S) orelse j=(S,R)) then P2
  else E
end );
```

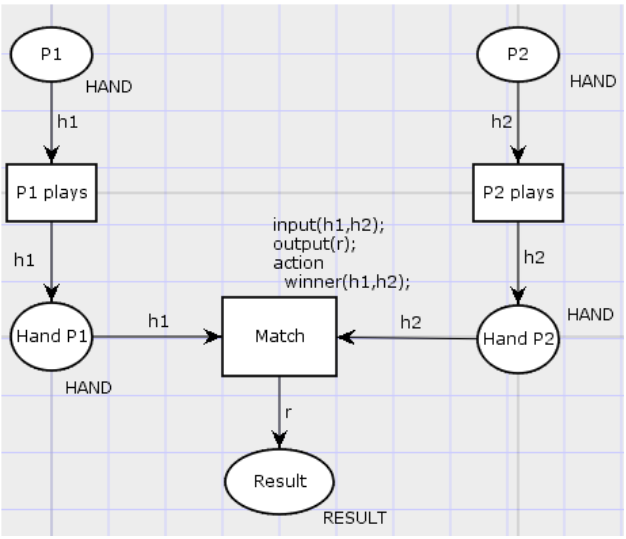

Figura 15: Modelo A para *Jokenpo*

O modelo apresentado na Figura 15 contém a representação do jogo, ainda livre da marcação inicial com *tokens*. Para fins de identificação, esse é o modelo A. Nele, *P1* e *P2* são locais que representam as jogadas que podem ser realizadas pelos jogadores. As transições *P1 plays* e *P2 plays* representam a escolha da jogada feita pelo jogador. *Hand P1* e *Hand P2* contém a jogada escolhida por cada jogador, e que são as entradas para a transição *Match*.

A função *winner*, presente em *Match*, contém a regra do jogo responsável por definir o jogador vitorioso. Dada duas jogadas, esta função indica qual jogador venceu a partida ou se houve empate.

Esse modelo poderia ser simplificado retirando os locais *P1* e *P2* e as transições *P1 plays* e *P2 plays*. Eles estão presentes para que a simulação do jogo possa ser controlada de forma a permitir escolher as jogadas que serão realizadas pelos jogadores. A Figura 16 apresenta o estado inicial e final de uma simulação de dez partidas. As marcações iniciais estão preenchidas de tal modo que o jogador 1 irá realizar cinco jogadas de Pedra, três de Papel, duas de Tesoura. O jogador 2, por sua vez irá realizar duas jogadas de Pedra, quatro de Papel e quatro de Tesoura. Essas jogadas correspondem aos recursos disponíveis por cada jogador e foram concebidas arbitrariamente pelo autor.

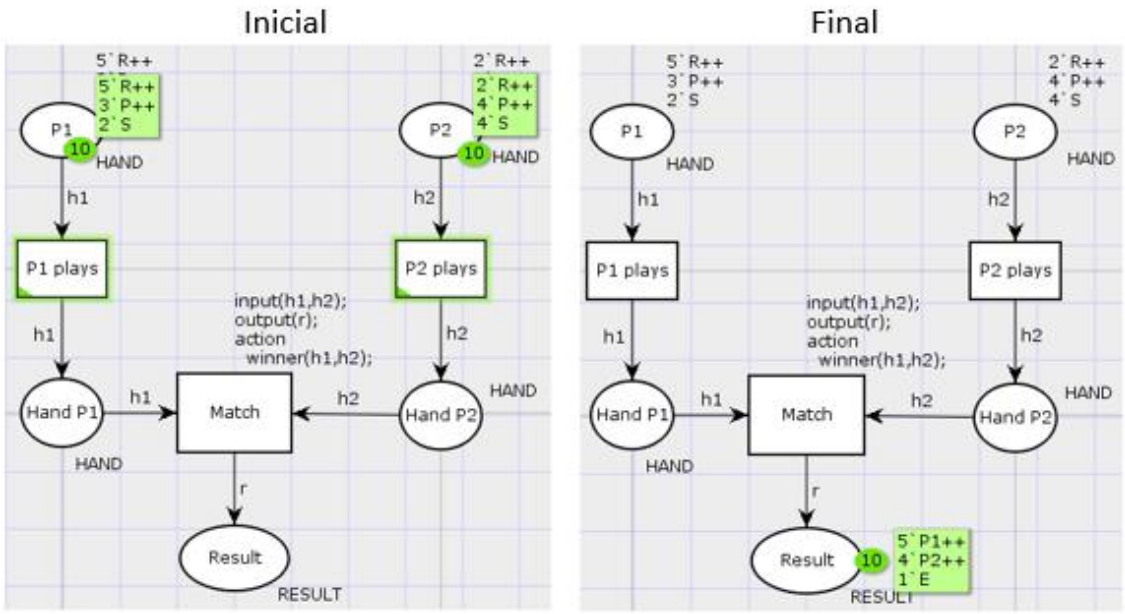

Figura 16: Modelo A para *Jokenpo*. Estado inicial e final

O estado final indica que a simulação resultou em cinco vitórias do jogador 1, quatro vitórias do jogador 2 e um empate.

Os recursos de um local são consumidos de forma arbitrária pela transição. Sendo assim, podemos observar que simulações de um mesmo modelo de jogo, com as mesmas marcações iniciais, podem resultar em estados finais diferentes. A figura a seguir apresenta os estados finais de outras simulações realizadas consecutivamente sobre o modelo A.

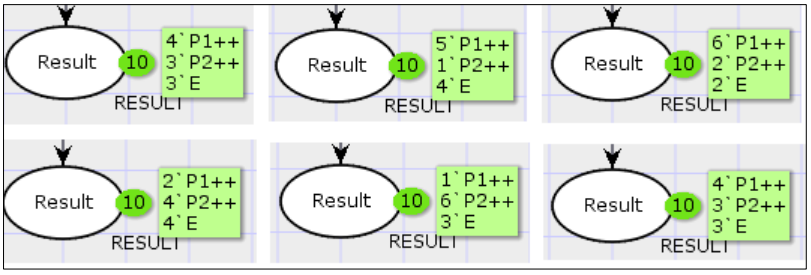

Figura 17: Modelo A para *Jokenpo* - estados finais

Por se tratar de um jogo de *Jokenpo* é esperado que, para um grande número de partidas realizadas, os jogadores tenham um número de vitórias bem próximo um do outro. Para verificar essa afirmação foi construído o modelo da figura a seguir, identificado como B.

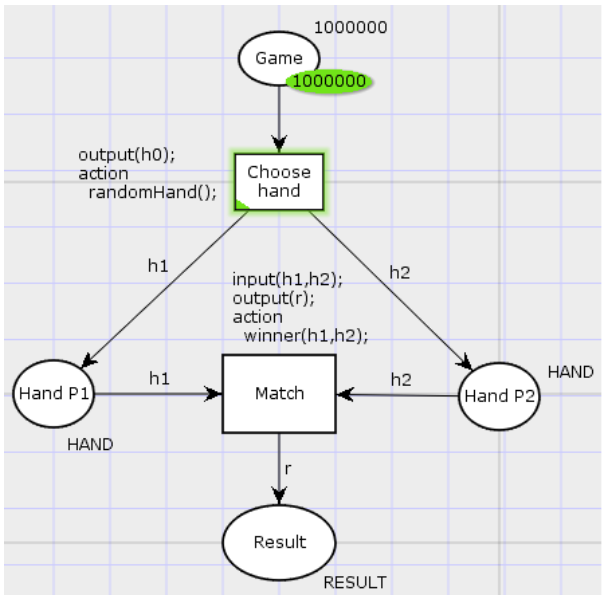

Figura 18: Modelo B para *Jokenpo*

O modelo B não possui os locais *P1* e *P2* e as transições *P1 plays* e *P2 plays*. Em seus lugares surgiram o local *Game*, com n *tokens* unitários com a finalidade única de disparar n vezes a execução do modelo. E a nova transição *Choose hand*, que por sua vez,

contém a função *randomHand*, responsável por gerar as jogadas que serão realizadas pelos jogadores.

As cores são as mesmas do modelo A. Foi adicionada a função *randomHand* para gerar aleatoriamente a jogada a partir de uma distribuição discreta uniforme. Outras funções de distribuições aleatórias também estão disponíveis no *CPN Tools* como, por exemplo, *bernoulli*, binomial, exponencial, normal e outras

A quantidade de *tokens* no local *Game* indica a quantidade de partidas disputadas na simulação. Nesse exemplo, assumimos que um milhão de partidas seria um número considerado grande. A execução de todas as partidas levou aproximadamente 11,5 segundos e o resultado foi o número de vitórias de cada jogador e empate muito próximos, conforme esperado matematicamente para esse tipo de jogo. A Figura 19 apresenta o estado final e o resultado consolidado.

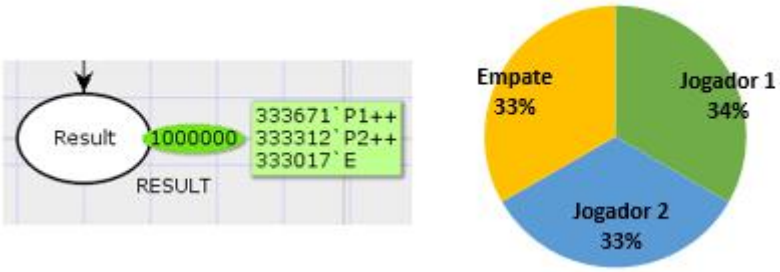

Figura 19: Estado final e resultado do modelo B para *Jokenpo*

3.1.1 Criando variações do jogo

O *Jokenpo* possui outras variações de jogo, ou extensões, que adicionam novas escolhas de jogadas e mantém a igualdade nas chances de vitória entre as possíveis jogadas. Tais extensões têm como ponto positivo o fato de que, quanto mais elementos estiverem disponíveis para serem jogados, menor será a probabilidade de uma partida resultar em empate.

Para avaliar a complexidade de realizar modificações na rede criada, a fim de construir uma variação deste, foi construída a rede correspondente para uma variação do *Jokenpo* conhecida como *Pedra-Papel-Tesoura-Lagarto-Spock* (PPTLS) (KAAS, 2012). Esta variação foi provavelmente criada na década de 1990, mas só ficou conhecida popularmente em 2008 ao aparecer no seriado americano *The Big Bang Theory* (CBS, 2015). Nesse jogo os elementos Lagarto e *Spock* são adicionados ao jogo de acordo com as seguintes regras: “Lagarto envenena *Spock* e come Papel”, “Lagarto é esmagado por

Pedra e decapitado por Tesoura", "*Spock* quebra Tesoura e vaporiza Pedra", "*Spock* é envenenado por Lagarto e refutado por Papel". A figura a seguir apresenta os resultados das possíveis jogadas.

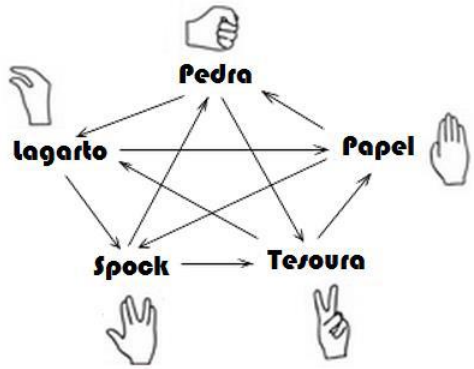

Figura 20: As regras de PPTLS. Adaptado de (KAAS, 2012)

Como a mecânica do jogo é a mesma, não foi necessário realizar alterações nas posições, transições ou arcos, de maneira que o modelo continua sendo o mesmo apresentado na Figura 18.

O *colorset* sofreu modificações. A cor HAND passou a ter duas novas possibilidades: L para Lagarto e V para *Spock*. A função *randomHand* passou a considerar esses dois novos elementos na geração da jogada escolhida pelo jogador e, por fim, a função *winner* recebeu as novas possibilidades de resultados de partidas, de acordo com as regras expostas na Figura 20

A execução de um milhão de partidas desse novo modelo levou aproximadamente 11,5 segundos. Conforme esperado, foi obtido um número de empates bem abaixo do número de vitórias de cada jogador, que por sua vez ficaram próximos. A Figura 21 apresenta o estado final e o resultado consolidado.

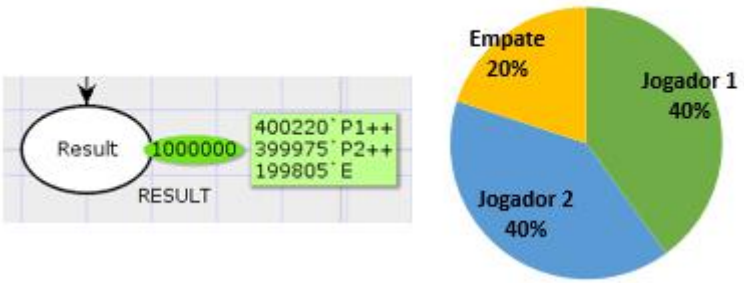

Figura 21: Estado final e resultado para PPTLS

As simulações descritas foram executadas dentro do ambiente do *CPN Tools*. Apesar de possuir limitações, o componente nativo de simulação se mostrou suficiente para os experimentos iniciais. Contudo, existem ferramentas de apoio a simulações no *CPN Tools* como *Petri Net Kernel* (PNK) (WEBER & KINDLER, 2003), *BRITNeY Suite* (WESTERGAARD & LASSEN, 2006) e *Access/CPN* (WESTERGAARD & KRISTENSEN, 2009), que podem auxiliar o desenvolvimento e fornecer resultados mais completos.

3.1.2 Exemplo mais complexo e suas dificuldades

Uma vez confirmada a viabilidade de utilização de Redes de Petri para a representação e simulação da economia de um jogo simples, foi construído o modelo de um jogo com mais complexidade.

O modelo apresentado na Figura 22 representa o jogo de cartas conhecido como Rouba-monte. Trata-se de um jogo que pode ser jogado por 2 ou mais jogadores e utiliza um baralho comum de 52 cartas. Neste jogo, cada jogador inicia com 4 cartas na mão e a mesa inicialmente também possui 4 cartas. Durante sua vez de jogar, o jogador deve verificar se alguma de suas cartas na mão possui o mesmo número de alguma carta na mesa ou do topo do monte de algum jogador. Caso possua, o jogador tem a opção de escolher se rouba a carta da mesa ou todo o monte do jogador. A jogada se dá colocando a carta de mesmo número sobre a outra e, posteriormente, colocando-as sobre seu próprio monte, aumentando-o. Caso o jogador não possua nenhuma carta igual a alguma das cartas sobre a mesa ou do topo do monte de algum jogador, então ele é obrigado a descartar uma de suas cartas sobre a mesa. Imediatamente, a vez passa para outro jogador. Após 4 rodadas os jogadores ficam sem cartas na mão e as cartas são distribuídas novamente e assim ocorre até que todas as cartas sejam distribuídas e os jogadores fiquem pela última vez sem cartas na mão. O vencedor será o jogador que terminar a partida com o maior monte de cartas (MEGAJOGOS, 2015).

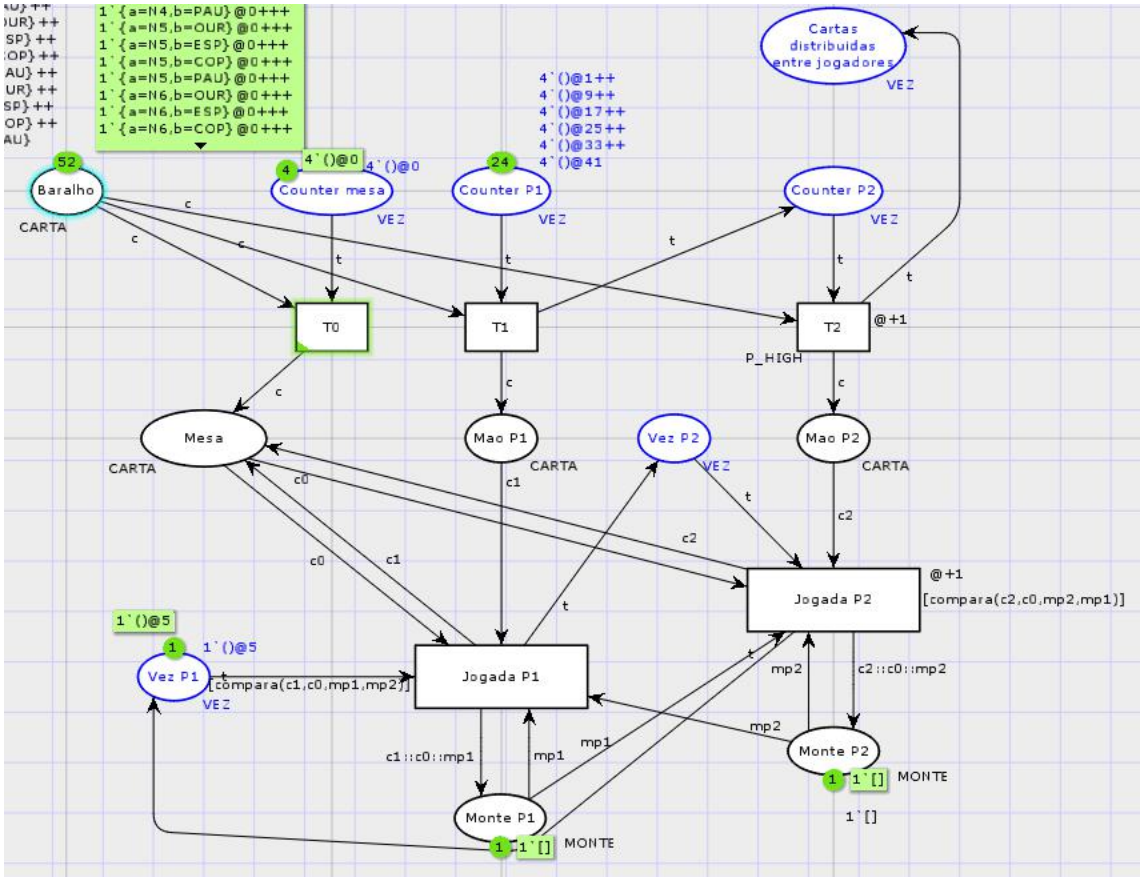

Figura 22: Modelo para Rouba-monte

É um jogo simples e fácil de jogar, porém introduz novos elementos que não existiam no modelo do *Jokenpo*, como:

- Cartas;
- Possibilidade de mais de dois jogadores;
- Controle da vez de cada jogador;
- Escolha entre diferentes ações na vez de jogar;
- Nem todos os recursos (cartas) estão disponíveis o tempo todo.

O modelo apresentado poderia ser mais simples, porém não iria aproveitar o uso das Redes de Petri de Alto-Nível, que podem conceder uma representação mais fidedigna à simulação do modelo. Para este modelo foram criados *colorsets* específicos que podem ser utilizados em qualquer jogo de cartas que utiliza o baralho convencional. Os *colorsets* são detalhados a seguir:

```
colset NUMERO = with AS|N2|N3|N4|N5|N6|N7|N8|N9|N10|J|Q|K;
colset NAIPE = with OUR|ESP|COP|PAU;
colset CARTA = record a:NUMERO * b:NAIPE;
colset MONTE = list CARTA;
```

Uma das dificuldades encontradas durante a criação do modelo foi o controle de turnos entre os jogadores e a distribuição periódica de cartas a cada 4 turnos. Outro problema encontrado foi a quantidade de jogadores. O modelo da Figura 22 é limitado e representa um jogo para somente 2 jogadores. Aumentar o número de jogadores significa aumentar ainda mais o modelo, replicando um conjunto de elementos do modelo.

O modelo construído não é tão fácil de ser compreendido ou sequer de ser concebido pelo *designer* do jogo que, provavelmente, não detém conhecimentos acerca de Redes de Petri. Um jogo mais complexo iria resultar numa representação mais complexa ainda. Esbarrar nessas limitações foram determinantes para abandonar essa tecnologia nesse trabalho.

3.2 Experiência com Máquinas de Estado

Uma das experiências anteriores de pesquisa desta tese envolveu a utilização de componentes UML para representação de jogos. A ideia era utilizar diferentes diagramas UML para criar representação de parte de um jogo e, então, aplicar alguma ferramenta de desenvolvimento orientada a modelo para gerar código de jogo.

Máquinas de Estado Finito (MEF) são modelos computacionais usados para representar uma quantidade limitada de estados de um sistema. As transições são responsáveis por alterar o estado atual da máquina e são disparadas por condições e eventos. A Figura 23 ilustra um exemplo de funcionamento de MEF. Quando se está no estado S1, o evento “a” leva a máquina ao estado S2. Quando se está no estado S2, o evento “a” não causa nenhuma alteração, e o evento “b” leva a máquina de volta ao estado S1.

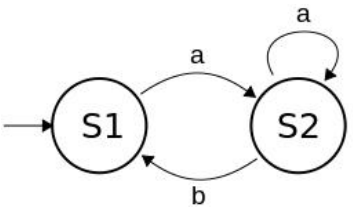

Figura 23: Exemplo de Máquina de Estado Finito simples

Não existe uma linguagem padrão para representar MEF, o que permitiu o surgimento de muitas ferramentas, como *Active HDL*, *Cinderella*, *Rational Rose* e também formatos distintos, como VHDL, SDL, UML, XML.

3.2.1 Máquinas de Estado e jogos

Modelos baseados em regras podem descrever conjuntos de condições que, uma vez satisfeitas, engatilham ações. De acordo com Fairclough et al. (FAIRCLOUGH *et al.*, 2001), Máquinas de Estado Finito se comportam como modelos baseados em regra.

Na literatura, é possível encontrar a aplicação de MEF em jogos para representar o comportamento de personagens não jogáveis (NPC, do original *non-player character*) (ROLLINGS & MORRIS, 2004). A Figura 24 apresenta o equivalente a um diagrama de MEF para representar o comportamento dos fantasmas inimigos do *Pac-Man*. As imagens de fantasmas foram adicionadas com fim puramente ilustrativo.

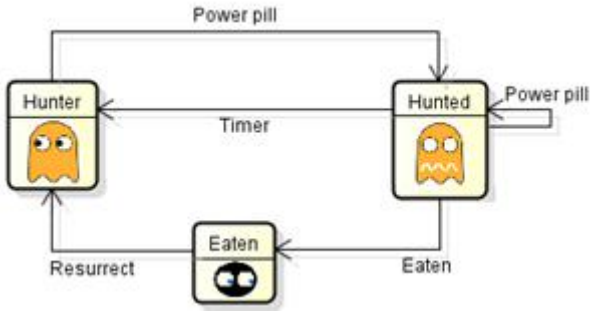

Figura 24: MEF para fantasma de *Pac-Man*. Adaptado de (ROLLINGS & MORRIS, 2004)

Nesse diagrama, a MEF é utilizada unicamente para representar os possíveis estados do fantasma.

Nelson e Mateas (NELSON & MATEAS, 2007) propuseram uma abordagem para formalizar mecânicas de jogo e gerar outros mini jogos. Em um nível inferior no processo de criação, Ashlock (ASHLOCK, 2010) apresenta um sistema de geração automática de *puzzles*, com diferentes níveis de dificuldade, para serem utilizados em conjunto com *design* de jogos.

Muitas ferramentas de criação no mercado utilizam modelos abstratos semelhantes a MEF, no sentido que possuem objetos que respondem a eventos de chamadas de funções, ou alguma abstração similar. No *GameMaker*, por exemplo, todo objeto segue um conjunto global de eventos pré-definidos que podem invocar uma função definida pelo programador. No *Construct 2*, há um conjunto de eventos, subeventos e comportamentos disponíveis para serem aplicados sobre os objetos. Os eventos são baseados em condições e podem disparar ações.

3.2.2 Geração de código a partir de MEF

Foi desenvolvido um mini-sistema para gerar código de jogo a partir de modelo de MEF. Os modelos deveriam ser construídos com alguma ferramenta de modelagem de máquina de estado capaz de exportar para o formato XMI, um formato padrão para troca de informações baseado em XML.

A Figura 25 apresenta uma visão do funcionamento do sistema.

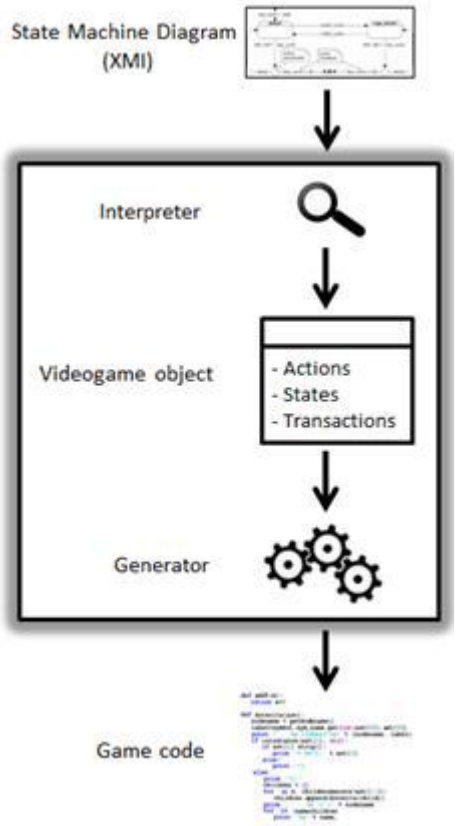

Figura 25: Sistema para geração de código a partir de MEF

O processo de desenvolvimento proposto compreendia três passos sequenciais:

- Modelo de MEF em XMI descrevendo estados e ações do objeto de jogo;
- Interpretação do modelo para obter as propriedades do objeto;
- Geração de código de jogo correspondente ao objeto modelado.

Foi implementado somente a geração de código correspondente às possíveis ações do personagem protagonista do jogo. Para isso, o diagrama MEF deveria ser construída com os estados representado as ações, e as transições como eventos disparados pelo jogador. Um conjunto de possibilidades de estados e transições foi previamente desenvolvido para que o gerador de código pudesse ler e capturar o elemento

correspondente. Os possíveis estados são: STAND; MOVE_UP, MOVE_DOWN, MOVE_LEFT, MOVE_RIGHT. E as possíveis transições são: HIT, PRESS, RELEASE, EXTERNAL.

A Figura 26 apresenta um exemplo de diagrama MEF para as ações do jogador em *Snake*, um popular jogo dos celulares dos anos 2000. Nesse exemplo, a cobra (avatar do jogador) está sempre se movendo em direção a uma das quatro direções possíveis, não podendo alterar o movimento para sua direção oposta. Sempre que o há uma colisão com um alimento a cobra aumenta de tamanho.

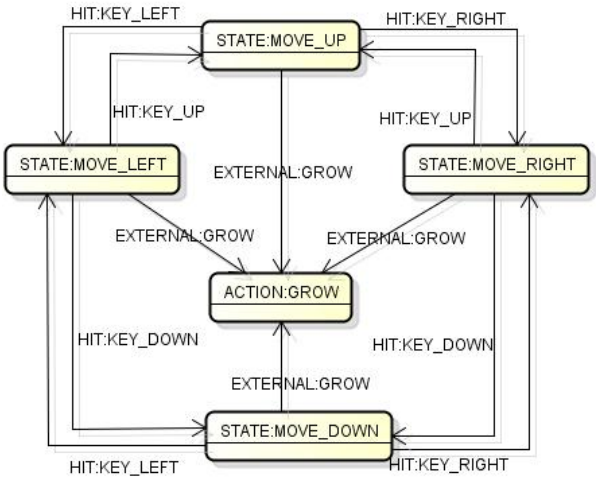

Figura 26: MEF para *Snake*, o popular "jogo da cobrinha"

O sistema de geração desenvolvido utiliza um esqueleto de código fixo, em Pygame, e incorpora o código gerado das possíveis ações do jogador, de acordo com o MEF que recebe de entrada.

Um outro exemplo construído usa o espaço como cenário e uma nave como avatar do jogador. O diagrama MEF construído para esse exemplo possui quatro estados para as direções em que a nave pode se mover e um para ficar parada. As transições alteram o estado ou fazem a nave atirar, nesse último caso o estado não é alterado. A Figura 27 apresenta o diagrama MEF e uma imagem com o cenário e nave do jogo, sendo possível movimentar a nave pelo cenário de acordo com os estados e transições do contemplados no diagrama.

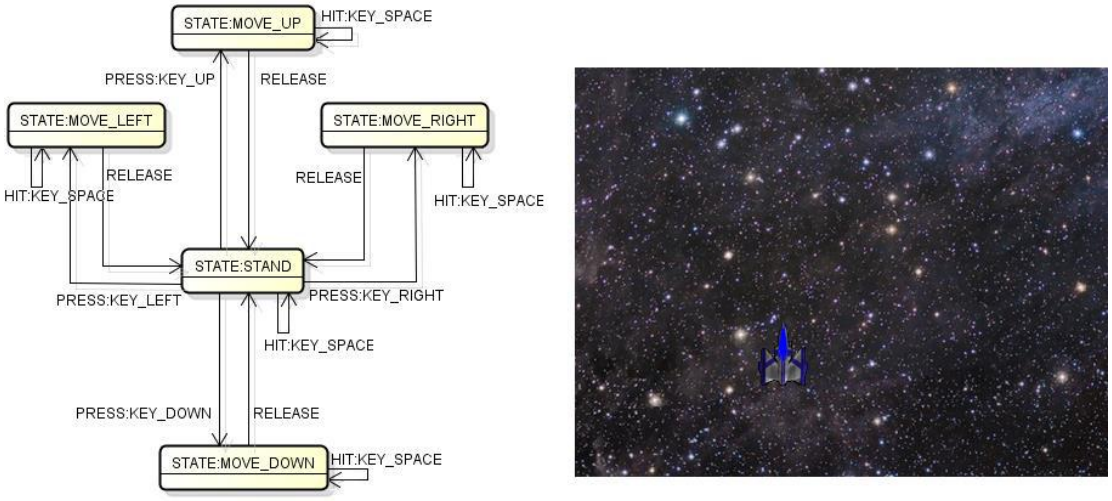

Figura 27: MEF para jogo de nave e imagem do jogo

O trabalho envolvendo máquinas de estado foi abandonado devido a várias limitações enfrentadas. Uma delas era a possibilidade de geração de código apenas para estados e transições pré-concebidas. Isso envolveria a criação de muitos estados e transições e, ainda assim, limitaria a criatividade do *designer* de jogo, deixando-o com opções restritas.

Outro problema era a dificuldade em criar ações mais complexas, por esse motivo somente ações básicas de movimentação foram desenvolvidas. Atirar, por exemplo, gerava um esqueleto de código com necessidade de posterior implementação manual.

As ações dos personagens constituíam uma pequena parte de um conjunto maior de representações pretendidas. Se nessa parte já foram encontradas tais limitações e dificuldades certamente haveriam novos problemas. Por conseguinte, o trabalho com máquina de estado se mostrou inviável para atender os propósitos desta tese.

4. Padrões para *Design* de Jogos

Padrões em jogos ajudam nas escolhas que surgem durante o processo de criação de um jogo, entendendo como outros jogos funcionam ou servindo como fonte de inspiração para novas ideias (BJÖRK & HOLOPAINEN, 2005). Exemplos de padrões incluem conceitos amplamente conhecidos como um chefe final para ser enfrentado no final da fase, *power-ups* para serem coletados, número limitado de vidas; além de outros conceitos abstratos como tensão da narrativa, liberdade de escolha de diferentes caminhos a seguir, possibilidade de enfrentar ou cooperar com outro jogador etc.

Björk e Holopainen criaram uma coleção de padrões para *design* de jogos com foco na interação dos jogadores com o jogo e entre si. Os primeiros padrões foram criados a partir da mecânica dos jogos e, posteriormente, aprimorados e incorporados a novos padrões que surgiram a partir de fundamentos teóricos, análises dos jogos e entrevistas com *designers* (BJÖRK & HOLOPAINEN, 2005).

Os padrões são classificados em onze categorias distintas. Cada padrão é apresentado pela sua definição, descrição, instruções para aplicação, consequências e relações com outros padrões. As onze categorias criadas por Björk e Holopainen são as seguintes:

- Padrões para elementos de jogos: correspondem aos componentes de um jogo, geralmente objetos, que os jogadores podem manipular e possuem características usadas para definir a área de realidade do jogo.
- Padrões para recursos e gerenciamento de recursos: quais são os diferentes tipos de recursos e como jogadores e o sistema de jogo podem controlar o fluxo dos recursos dentro e fora do jogo.
- Padrões para informação, comunicação e apresentação: diz respeito a como a informação sobre o estado do jogo está disponível para os jogadores ou oculta deles. São padrões para controlar a maneira que o jogador tem acesso a informações dos elementos de jogo, mas também pode ser usado para esconder determinadas ações executadas ou objetivos de seus adversários.
- Padrões para ações e eventos: que tipos de ações estão disponíveis para os jogadores, como elas se relacionam com as mudanças no estado do jogo e com os objetivos dos jogadores.

- Padrões para estruturas de narrativas, previsibilidade e imersão: maneiras de suportar imersão, controlar a narrativa do jogo e conquistar o comprometimento dos jogadores.
- Padrões para interação social: cobrem as principais possibilidades da disponibilização de interação entre os jogadores. Existem jogos em que os jogadores são encorajados a interagirem blefando, enganando; já em outros incentivam a colaboração.
- Padrões para objetivos: descreve vários formatos específicos de definir objetivos e como eles afetam o *gameplay*.
- Padrões para estruturas de objetivos: são padrões que descrevem aspectos do jogo relacionados aos objetivos, mas não são os objetivos. Inclui padrões de como atingir os objetivos podem fazer um jogador vencer o jogo, como os objetivos se relacionam entre si e como eles podem afetar as relações dos jogadores.
- Padrões para sessões de jogo: correspondem a características de quando os jogos são jogados de fato. Inclui limitações, possibilidades, e funcionalidades da participação do jogador.
- Padrões para domínio de jogo e balanceamento: descrevem algumas características importantes relacionadas à forma como os jogadores podem usar suas habilidades no jogo e a possibilidade de equilibrar o *gameplay* para jogadores com diferentes habilidades.
- Padrões para meta-jogos, *replayability* e curva de aprendizado: descrevem recursos para apoiar participação de jogadores fora do jogo.

Estes padrões forneceram fundamento teórico de grande valia e foram utilizados na elaboração da proposta de tese e no simulador construído. Se destacam principalmente os padrões para elementos de jogos; recursos e gerenciamento de recursos; ações e eventos; objetivos; sessões de jogo; e domínio de jogo e balanceamento. As seções desse capítulo abordam maiores detalhes dos padrões de (BJÖRK & HOLOPAINEN, 2005) contemplados na ferramenta de simulação construída.

4.1 Avatar

Avatar é um padrão para elemento de jogo fortemente conectado ao sucesso ou falha do jogador. Em muitos casos é o único elemento que pode ser controlado pelo

jogador, como a Lara Croft da série de jogos *Tomb Raider* (Figura 28). O *design* do *avatar* é principalmente baseado em que ações de *gameplay* ele pode executar.

Figura 28: Lara Croft em Rise of the Tomb Raider

Uma construção muito utilizada é a evolução conforme se obtém progresso no jogo, podendo adquirir novas habilidades ou ações.

As ações do *avatar* podem ser realizadas de acordo com os atributos que possui. O grau de impacto de suas ações está relacionado com seus atributos. Geralmente, a morte ou destruição do *avatar* pode significar o fim do jogo, a diminuição gradual ou completa de algum de seus atributos.

O *avatar* também é um instrumento para aumentar a empatia do jogador com o jogo, fazendo-o sentir-se o próprio. Em jogos com câmera de primeira pessoa o fator imersão é acentuado, porém o *avatar* raramente aparece. Já a visão em terceira pessoa favorece o *avatar* e o torna mais suscetível a conseguir "conquistar" o jogador.

Esse padrão fundamenta o artefato *personagem* do simulador.

4.2 Unidades

São grupos de elementos sob o controle do jogador, podem ter diferentes ações e atributos associados. Um único jogador pode controlar diversas *unidades* e perder uma *unidade* não necessariamente significa que perdeu o jogo. Assim como *avatar*, também é um padrão para elemento de jogo.

A quantidade de *unidades* disponíveis para um jogador deve ser limitada, assim como os recursos das *unidades* como um todo e de cada uma de suas partes individuais. Existem tipos de jogos em que novas *unidades* podem ser conquistadas durante o

progresso do jogo e, então, passam a integrar o conjunto de *unidades* disponíveis ao jogador. Também é possível ao jogador controlar separadamente vários grupos distintos de *unidades,* com particularidades e características próprias.

Uma das dificuldades do *design* de *unidades* é obter um balanceamento satisfatório de seus atributos de acordo com a intenção do jogo: ser fácil, médio ou difícil.

Esse padrão fundamenta o artefato *time* do simulador.

4.3 Inimigos e Chefões

São dois padrões para elementos de jogos muito parecidos. Os *inimigos* podem possuir as mesmas características de *avatar* ou de *unidades*, porém com a finalidade de impedir que os jogadores avancem no jogo.

Há muitas maneiras de atrapalhar o avanço dos jogadores, como a utilização de labirintos, armadilhas, eventos aleatórios que prejudicam o jogador, etc. Os *inimigos*, de maneira geral, se movimentam e procuram destruir o jogador. Para superar o *inimigo* o jogador precisa confrontá-lo até que seja derrotado ou, em alguns jogos, há a possibilidade de escapar do confronto ou evitar seguindo por outro caminho.

Como os *inimigos* podem estar na forma de unidades, vale ressaltar novamente a importância do balanceamento para uma boa experiência de jogo.

O *chefão* é uma especialização do inimigo mais poderosa. Há jogos em que só há como progredir depois de superar o *chefão*. Ele também pode ser uma espécie de guardião de um local que contém recompensas que podem facilitar o progresso do jogador. É comum em jogos de progressão existirem vários *inimigos* no decorrer das fases e um *chefão* no final, marcando que a fase termina ali. Existem *chefões* que à primeira vista parecem indestrutíveis, mas possuem um ponto fraco. O jogador deve utilizar de estratégias e realizar ações a ponto de conseguir atingir tal ponto para derrotar o *chefão*.

Esses padrões fundamentam o artefato *inimigo* do simulador.

4.4 Itens, *pick-ups*, *power-ups*

Itens e os demais elementos de jogo que os utilizam são entidades separadas e podem existir independentemente um do outro. Em geral, *itens* podem ser coletados, descartados, destruídos, negociados e assim por diante. Assim sendo, eles não têm uma posição fixa no *mundo de jogo*.

Armas, armaduras, poções, chaves e outros são exemplos de *itens* frequentemente encontrados em jogos. O principal fato sobre um *item* é a ação que o acompanha, como a melhoria de determinada habilidade ou possibilidade de acessar alguma área do jogo.

Os *itens*, em geral, são elementos *pick-ups*, ou seja, podem ser coletados durante o jogo. Também existem recursos que não se enquadram como item, mas que são pick-ups, como por exemplo comida e dinheiro.

Os *power-ups* são tipos especiais de *pick-ups*. Tratam-se de elementos de jogo que agraciam com alguma vantagem o *personagem* que o coleta. Geralmente conferem melhoria de algum *recurso* ou habilidade limitado por tempo ou por quantidade de uso.

Esses padrões fundamentam o artefato *item* do simulador.

4.5 Mundo de jogo e Mundo configurável

O *mundo de jogo* é o ambiente onde ocorre o *gameplay* e as relações espaciais entre os elementos de jogo. Este padrão determina os tipos movimentos e ações que podem ser desempenhadas.

Normalmente é um espaço concreto e limitado em uma dimensão como o jogo *Gamão,* por exemplo; duas dimensões como em *Pac-Man;* ou três dimensões como a maioria dos jogos digitais mais modernos. Existe ainda a opção de *mundo de jogo* abstrato e construído pelos próprios jogadores, como ocorre em jogos de contar histórias.

O padrão de elemento *mundo configurável* remete a jogos em que os jogadores podem criar ou alterar de alguma forma o mundo. Em *Super Mario Maker* o jogador pode criar seu próprio jogo com os elementos tradicionais do mundo de Mario. Os jogos da série Grand Theft Auto permitem modificações (*mods*) que alteram a física do jogo. Também se encaixam nesse padrão os jogos que oferecem ao jogador a opção de selecionar o nível de dificuldade.

Esse padrão fundamenta o artefato *mundo* do simulador.

4.6 Level

Os *levels* são parte do *mundo de jogo* no qual os jogadores estão presentes até que um certo objetivo seja alcançado ou passem para outro *level*, devido a alguma condição de jogo.

A diferença entre os *levels* de um jogo pode ser no seu conteúdo, estética ou combinação de ambos. Em alguns jogos os *levels* se diferenciam somente pelo aumento

progressivo da dificuldade, em outros há variações de *inimigos* e elementos de jogo fazendo com que um *level* seja completamente diferente do anterior. Não é raro encontrar jogo com um ou outro *level* sendo utilizado para adicionar elementos de outro gênero de jogo. Essa adição de gênero deve ser realizada com cautela para que a experiência não seja desastrosa e frustrante para o jogador, como em *Battletoads* (Figura 29), um jogo de ação de plataforma que possui um *level* extremamente difícil em que o jogador só pode controlar uma moto espacial desviando de obstáculos.

Jogos com estrutura de progressão frequentemente contém *levels* explicitamente numerados, com possibilidade ou não de retornar ao *level* anterior. Em jogos com essa estrutura muitas vezes o objetivo é chegar ao final do *level*. Há também jogos em que os *levels* são apenas separações de espaços dos elementos que ali existem, como em jogos de mundo aberto cujo objetivo é derrotar um *chefão* específico ou encontrar algum objeto escondido, não sendo necessariamente obrigatório percorrer por todos os *levels* para que tal objetivo seja atingido.

Esse padrão fundamenta o artefato *local* ou agrupamento de locais, do simulador.

Figura 29: *Battletoads* e a parte da moto espacial (RARE, 1991)

4.7 Áreas inacessíveis e Obstáculos

Áreas inacessíveis são partes do *mundo de jogo* que o *personagem* nota sua existência, mas não consegue entrar por algum impedimento. A área pode se manter constantemente inacessível, como o gerador de fantasmas de *Pac-Man*, ou ser acessível após se livrar do *obstáculo* que a impede de ser adentrada.

Os *personagens* podem interagir com os *obstáculos*, por vezes destruindo-o, ou movendo-o, ou utilizando neles o *item* correto, visando superá-los.

Esses padrões fundamentam o conceito de *chave e cadeado,* que pode ser empregado em *locais*.

4.8 Recursos

Este padrão é o principal da categoria de padrões para recursos e gerenciamento de recursos, diferente dos padrões para elementos de jogo apresentados anteriormente. Pode-se definir *recursos* como elementos que permitem aos jogadores realizarem determinadas ações no jogo.

Recursos podem ser representados como elementos físicos ou virtuais. Em jogos digitais os recursos mais comuns são vida, energia, dinheiro, munição, força, poder e outros, dependendo do jogo. Quando um *designer* constrói um *recurso* sua principal pergunta é: “qual seu propósito?”. Existem vários usos diferentes para os recursos. A seguir os principais:

- Como elemento de troca: *recursos* que podem ser utilizados para trocar por outros *recursos* que irão conceder melhores vantagens ao jogador.
- Como meio de executar ação: o jogador precisa gastar o recurso para executar a ação, como munição para atirar com determinada arma; ponto de poder para lançar magia; ponto de cura para ganhar vida.
- Como fator de durabilidade: o tempo pode ser aplicado como *recurso* de esgotamento automático para marcar o período em que uma habilidade vai estar ativa; ou prazo para concluir o *level*. Já *recursos* de esgotamento condicional são consumidos em decorrência de eventos, tal como a vida do personagem para indicar que ele permanece ativo no jogo; escudo ou armadura de proteção que vão se enfraquecendo conforme são atingidos.

Como pode ser notado, os atributos que compõem o *avatar* ou *unidades* do jogo são *recursos*. Em jogos cooperativos ou com escolha de suas *unidades* é interessante que haja distribuição dos *recursos* de forma que quando um *recurso* é destacado outros são preteridos. Assim é evitado que um jogador se torne muito mais poderoso que outro.

Esse padrão fundamenta os atributos de *personagem*, e controle de unidade de tempo do simulador.

4.9 Limite de ações, de tempo e de recursos

Existem alguns padrões que tratam de limites. O *limite de ações* define quantidades limitadas de opções e de números de escolhas a serem feitas em um determinado período de tempo ou turno.

A quantidade de ações disponíveis pode ser fixa durante todo o jogo ou estar condicionada a algum fator variável, como o nível de poder de determinado *personagem* no momento. Em jogos de turno, sem *limite de tempo*, o número de ações pode ser determinante para a condição de encerramento de turno. Quando há *limite de tempo* um jogador pode demorar muito e ver seu turno se encerrar antes de executar todas as ações possíveis.

O *limite de recursos* faz com o que os jogadores planejem as ações que irão executar para não ficarem sem determinado *recurso*. Apesar de limitados, recursos podem ser automaticamente renováveis ou renovados pelo uso de *itens* coletados pelo *mundo de jogo*.

O uso de limites nos jogos, por um lado diminui a liberdade de escolha do jogador, mas por outro lado força o planejamento e estratégia, obrigando os jogadores a pensarem mais atentamente antes de agir.

Esses padrões fundamentam o mecanismo de batalha, e como os recursos são consumidos.

4.10 Objetivos predefinidos e opcionais

Os *objetivos predefinidos* são estabelecidos pelo *designer* do jogo, geralmente dispostos em uma hierarquia rígida, que só podem ser adaptáveis pelas escolhas ou interpretações dos jogadores se o design o permitir. Todos os jogos que possuem um ou mais vencedores possuem o objetivo primário predefinido: vencer. Pode parecer banal já que a maioria dos jogos possui vencedores, mas ainda assim existem jogos como *Tetris*, *Enduro* e os da série *The Sims*, em que não existem condições que definem a vitória do jogador.

Para se atingir o objetivo principal de vencer o jogo, por vezes é necessário cumprir uma série de outros *objetivos predefinidos* menores. O *designer* de jogo pode estipular se eles devem ser cumpridos em uma ordem específico ou não.

Os *objetivos opcionais* são metas que o jogador não precisa necessariamente cumprir para finalizar o jogo com sucesso. Geralmente há alguma *recompensa* para estimular o jogador a cumprir tais objetivos, como fornecer algum *item* que torna o personagem mais poderoso, ou apenas entreter de alguma maneira.

Ao considerar criar um *objetivo opcional* é preciso analisar se o objetivo é realmente opcional ou se ele é pré-requisito para um *objetivo predefinido*. Por exemplo, um monstro a se enfrentar no caminho principal do jogo, mas que só pode ser derrotado com uma arma obtida num caminho opcional. Nesse caso, passar por esse caminho passa a ser obrigatório para o prosseguimento do jogo e, portanto, este também é um *objetivo predefinido*.

Outra questão é ponderar se deve ser claro ou não para o jogador que ele se encontra no caminho de um *objetivo opcional*. Isto é comum quando os *objetivos opcionais* não fornecem nenhum tipo de recompensa e são utilizados somente como distração ao jogador para que ele demore a encontrar o caminho principal que o leva aos *objetivos predefinidos*.

Esses padrões fundamentam a disposição dos elementos pelo *mundo de jogo* elaborado pelo *designer*.

4.11 Movimentação e Travessia

Como o próprio nome já define, o padrão de *movimentação* representa a ação de mover os elementos pelo *mundo de jogo*. O *designer* de jogo define as possibilidades de *movimentação* de acordo com cada elemento e sua intenção com o jogo. Jogos de corrida de carro, por exemplo, possuem a *movimentação* como principal mecânica do jogo. No xadrez cada peça possui seus movimentos específicos.

A dificuldade de *movimentação* está diretamente ligada a liberdade de escolha do jogador, de forma que quanto mais possibilidades de movimentação existem maior se torna a dificuldade. Isso significa que deve haver um equilíbrio entre o nível de complexidade dos movimentos e da liberdade de escolha, para que os jogadores tenham uma boa experiência de jogo.

Travessia é objetivo de tentar mover um elemento de jogo de uma posição para outra. Jogos de plataforma como *Sonic, Altered Beast* e os da série *Super Mario* são exemplos em que o objetivo de cada *level* é levar o *personagem* do seu início até o fim. Esse padrão se faz muito presente em jogos de progressão e é tipicamente combinado

com outros padrões para criar variações e diferentes níveis de dificuldade, como incluir *inimigos* e *obstáculos* no caminho, criar caminhos alternativos, limitar o tempo para realizar a travessia.

O padrão *travessia* fundamenta parte dos objetivos de simulação.

4.12 Eliminação, Combate e Recompensas

Eliminação significa remover um ou mais elementos do *mundo de jogo*. O motivo da *eliminação* pode ser evitar alguma ameaça ao jogador, ou pode ser essencial para se atingir o objetivo de jogo. Os elementos de jogos não são necessariamente eliminados de forma permanente. Existem jogos onde após um certo tempo os elementos reaparecem em locais específicos do *mundo de jogo*. Em geral, a *eliminação* ocorre através de alguma ação dos *personagens* ou *unidades*, como por meio de *combates*.

Combate é o conflito entre *personagens* ou *unidades*, comumente terminando após a derrota de um dos lados. Este padrão é uma das formas mais antigas e comuns de associar o jogo com competição do mundo real entre os jogadores, além de fazer alusão a tensão e incerteza.

Os *combates* em jogos podem ser em tempo real ou em turnos. Ambos os casos geralmente incluem aleatoriedade e informações imperfeitas, de forma que não deve ser possível prever antecipadamente o lado vencedor.

Recompensas são os efeitos positivos que os jogadores esperam conseguir ao completarem objetivos. Constituem uma das principais formas pelas quais os *designers* de jogos têm de encorajar os jogadores a fazerem certas ações em um jogo. No entanto, os jogadores devem sentir que a *recompensa* pode melhorar suas chances no jogo ou fornecer alguma sensação agradável fora do jogo. Em *Zelda*, não é obrigatório procurar e coletar os corações, mas os jogadores procuram porque a *recompensa* é aumentar o nível de vida do *personagem*.

A *recompensa* também pode ser na forma de história, revelando detalhes sobre a história de algum *personagem* ou do jogo que não estão presentes no caminho principal. Há ainda jogos que exibem a porcentagem do jogo concluída. Pode ser frustrante para alguns jogadores finalizar o jogo e saber que não o completou por inteiro. Para tal, é necessário cumprir todas as missões e encontrar todos os coletáveis do jogo. Nesse caso, a recompensa é a satisfação em visualizar o desejado 100%.

Os padrões *eliminação* e *combate* fundamentam o sistema de batalhas do simulador. As *recompensas* se fazem presentes pela opção de incluir *itens*, que podem ser ganhos decorrente do sucesso em uma batalha.

4.13 Jogos baseados em turnos e Jogos baseados em tempo

Em *jogos baseados em turnos* os jogadores alternam turnos para realizar suas ações e o tempos de jogo não é contado com o tempo real. Muitas vezes existem jogadores que levam muito tempo para realizar as suas ações e passar a vez ao próximo jogador. Para não causar sensação de frustração nos demais, alguns jogos colocam um limite de tempo máximo para a duração de um turno. Esse artifício pode causar pressão sobre o jogador durante seu turno.

Em *jogos baseados em tempo*, o tempo de jogo progride acordo com o tempo real, mas em passos discretos. A passagem de tempo costuma ser medida em uma unidade atômica chamada *tick*. Cada ação desempenhada pelos personagens é medida em conjunto de *ticks* que leva para realizá-la, assim podem ter ações que levam mais tempo que outras. O atraso em tempo real entre os *ticks* no tempo de jogo pode variar de jogo para jogo.

Esses padrões fundamentam os modos do sistema de batalha do simulador.

4.14 Aleatoriedade e Dados

O padrão *aleatoriedade* define a existência de efeitos ou eventos imprevisíveis pelos jogadores. Isso não faz com que os jogos sejam totalmente imprevisíveis, pois deve haver uma estrutura onde os jogadores possam conhecer as chances de certos acontecimentos. Jogos com esse elemento fornecem tensão e conseguem afetar emocionalmente os jogadores.

A *aleatoriedade* também permite que os jogadores tenham uma chance perceptível de sucesso independente de qualquer habilidade, causando menos pressão sobre seu desempenho e sentimento de estar com sorte, ou azar. A dose de *aleatoriedade* deve ser medida com cautela para não desestimular os jogadores, que precisam ter a sensação que continuam no controle.

Jogar *dados* é uma prática comum para criar aleatoriedade, pois trata-se de um mecanismo para gerar número aleatório em que as chances de qualquer resultado são exatamente as mesmas que as anteriores.

Em jogos digitais, uma jogada de *dados* pode ser simulada pela geração aleatória de um número em um intervalo de pré-definido. O resultado pode ser manipulado para obter efeitos de equilíbrio do jogador com o jogo, porém os jogadores podem perceber ao longo do tempo e, portanto, esta prática deve ser evitada para não ocasionar o desinteresse pelo jogo.

Esses padrões fundamentam os bônus de ataque e defesa do sistema de batalhas do simulador.

4.15 Balanceamento e Nível de dificuldade

O *balanceamento* está conectado a noção de chances iguais para todos os jogadores. Em jogos competitivos, em geral os jogadores iniciam em posições simétricas para que todos tenham a mesma chance, embora costuma sair em leve vantagem o jogador que inicia. Há jogos de luta que permitem ao jogador aumentar ou diminuir a vida do personagem, para que jogadores com níveis de habilidade diferente possam jogar e se divertirem.

Em jogos para um jogador, o comportamento esperado é um jogo em que a dificuldade vai aumentando com o avançar das fases. Seria muito estranho um jogo onde a primeira fase é a mais difícil e o restante do jogo muito fácil. Os inimigos e obstáculos precisam estar balanceados de forma que o jogo tenha essa harmonia.

Acertar a dificuldade de um jogo pode ser uma tarefa difícil devido a subjetividade das habilidades dos jogadores. Uma alternativa para atingir todos os públicos é permitir a seleção do *nível de dificuldade*. Isso significa que a força, intensidade e quantidade de inimigos e obstáculos do jogo podem variar de acordo com o bem entender do jogador. Em geral os níveis para seleção são fácil, médio e difícil. Contudo, também é comum existirem outros níveis além destes ou, ainda, modos mais difíceis que são liberados depois que o jogador finaliza o jogo no modo difícil.

Em *Duke Nukem* (Figura 30), a tela de seleção do *nível de dificuldade* é feita de forma bem-humorada utilizando frases no lugar do tradicional “fácil, médio, difícil”.

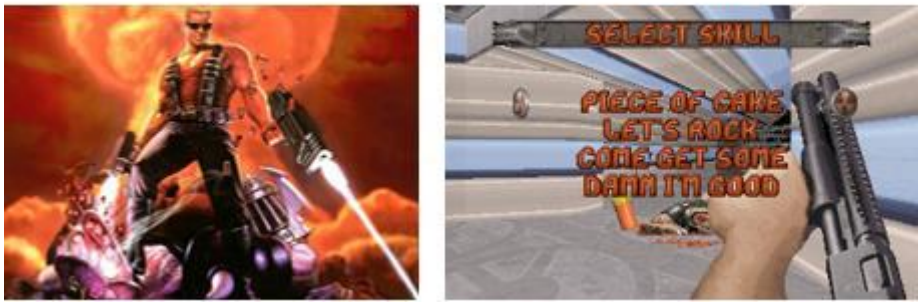

Figura 30: *Duke Nukem* e sua clássica tela de seleção de dificuldade

Esses padrões fundamentam parte dos objetivos do simulador, uma vez que o propósito dessa tese é auxiliar o *designer* tanto na elaboração de jogos, quanto no seu refinamento, com o intuito de atingir o balanceamento e nível ou níveis de dificuldade desejados.

5. Proposta de Ambiente de Simulação de Jogos de Progressão

Conforme apresentado anteriormente, ferramentas como *Machinations* se propõe a auxiliar o *designer* na elaboração de jogos emergentes. Esse capítulo descreve a proposta de um ambiente para auxiliar o *designer* na elaboração de jogos de progressão.

5.1 Delimitação de tipos de jogos

Após a análise de jogos de progressão e os elementos típicos que os compõem foi feita a escolha por uma determinada família de jogos, baseada nas características de balanceamento e dinamismo em se obter diferentes jogos a partir de variações de elementos.

Dessa forma, o ambiente proposto suporta jogos com elementos típicos dos gêneros *Role Playing Game (RPG)* e *Aventura*, que são gêneros tipicamente de progressão e contém grande quantidade de elementos que devem ser balanceados para que o jogo não seja muito fácil ou muito difícil.

É indiferente se o jogo é para um ou vários jogadores. Nesses estilos de jogos o jogador, ou jogadores, controlam um personagem ou uma equipe e iniciam uma jornada partindo de um ponto A até o ponto B. Durante a jornada, o jogador passa por diversos locais que podem conter inimigos, itens e obstáculos. Não há necessariamente um único caminho a ser percorrido, de forma que os locais visitados durante a jornada podem ser obrigatórios ou opcionais, resultando em inúmeras possibilidades de caminhos que levam ao local final.

Jogos no estilo *Dungeon Crawl*, como *Rogue*, *Gauntlet*, *Dungeonlike* e *Diablo* se encaixam muito bem nas características descritas. Além disso, um modelo de jogo com tais características é genérico o suficiente para atender jogos de outros gêneros que também possuam as mecânicas de percorrer locais, resolvendo possíveis conflitos em cada um deles, com o objetivo de chegar em um local final.

5.2 Descrição do ambiente de simulação

O ambiente é composto por um simulador, artefatos de entrada e artefatos de saída. O simulador funciona de maneira análoga a uma fábrica da época da revolução industrial, os artefatos de entrada são a matéria prima e os artefatos de saída correspondem ao produto final. O *designer* executa o papel de dono dessa fábrica, controlando todo o processo.

A Figura 31 fornece uma visão macro destacando papel dos artefatos de entrada para o simulador. *Mundo* e *configurações de simulação* compõem os artefatos de entrada. O *mundo* é constituído de um ou mais *locais*, que por sua vez podem conter *equipes* e *itens*. Cada *equipe* é formada por um ou mais *personagens*.

As *configurações de simulação* podem incluir diversas personalizações para que o *designer* de jogo tenha possibilidade de ajustar e testar diferentes cenários e combinações de configurações a fim de obter diferentes resultados. Os artefatos de saída são dados coletados de todos os eventos que ocorrem durante as simulações. É desejável que esses dados estejam em formato que proporcione fácil leitura, para possibilitar a geração de informações valiosas, relatórios, estatísticas, resumos e gráficos.

A ideia do ambiente é simular um grande número de partidas para gerar dados. Em cada simulação uma equipe de heróis parte de um local inicial, sendo a simulação finalizada quando chegam com sucesso ao local final, ou quando falham sendo derrotados por equipes de inimigos.

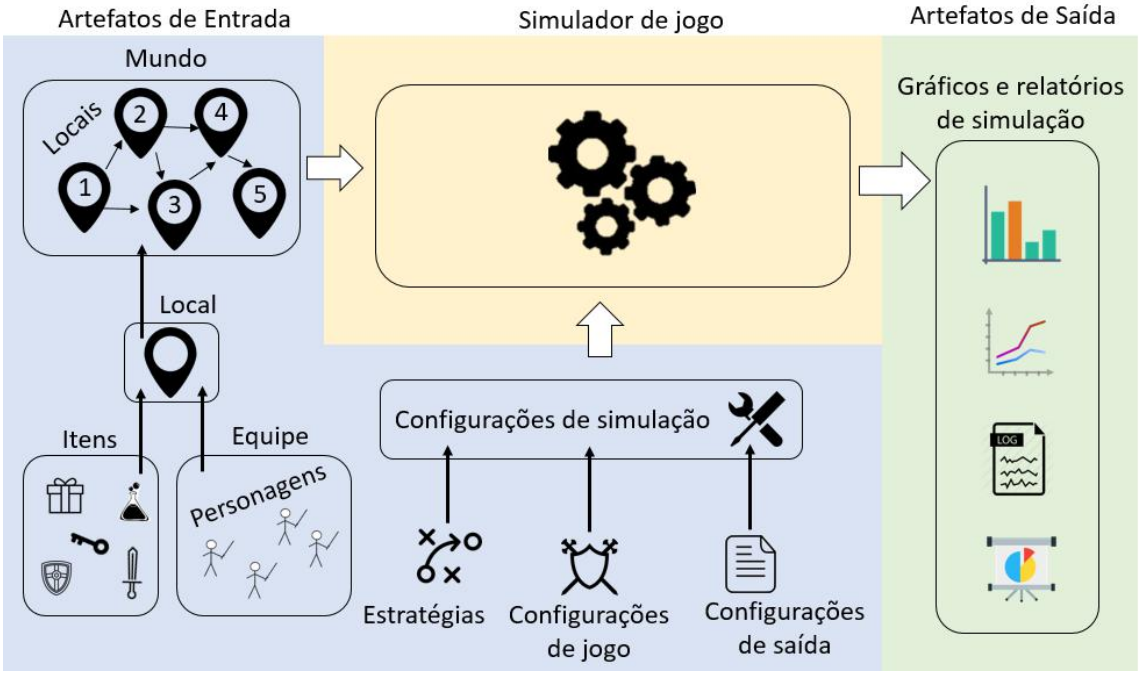

Figura 31: Ambiente de simulação

O diagrama de atividades da Figura 32 ilustra como o *designer* de jogo deve atuar no ambiente proposto. Sua inspiração continua fundamental em todo o processo. O *designer* deve construir a primeira versão dos artefatos de entrada para alimentar o simulador (*Design inicial*). A partir daí o *designer* inicia o *Ciclo de testes*, isto é, execução de simulações (*Simulações*), análise dos resultados (*Análise*) e, sempre que necessário, ajuste de artefatos e configurações para iniciar um novo ciclo (*Refinamento*). Uma vez que o *designer* aprova o resultado, a participação do simulador se encerra (*Fim*). Nesse momento, os elementos de *design* são considerados prontos para avançar no processo de construção de jogo.

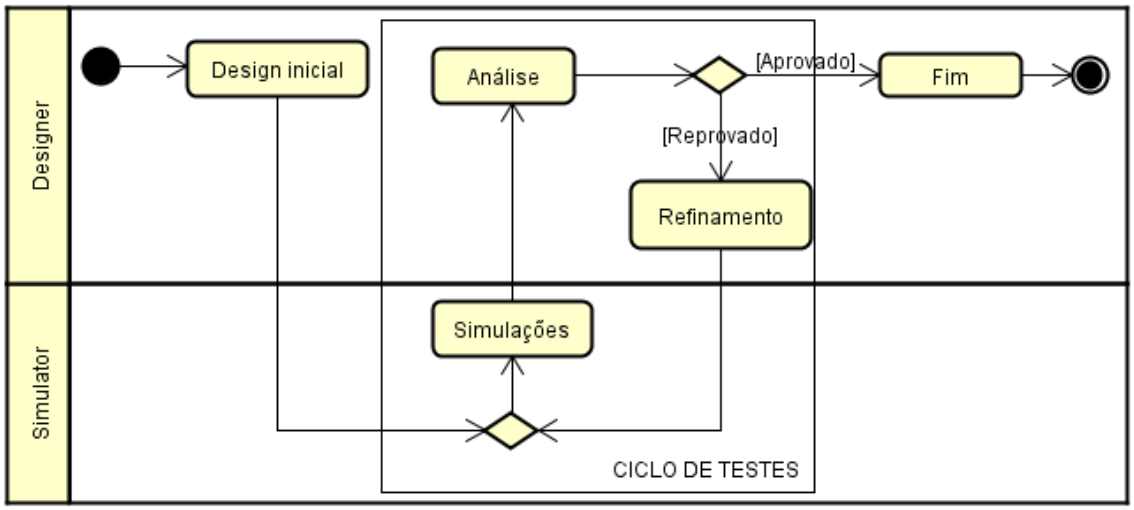

Figura 32: Atuação do *designer* de jogo

5.3 Requisitos

Essa seção descreve os requisitos necessários para o funcionamento esperado do ambiente proposto.

5.3.1 Artefatos de entrada

Além das configurações de simulação, os demais artefatos de entrada representam elementos fundamentais em jogos com as características da proposta. A Figura 33 apresenta um diagrama de classes com o relacionamento destes elementos.

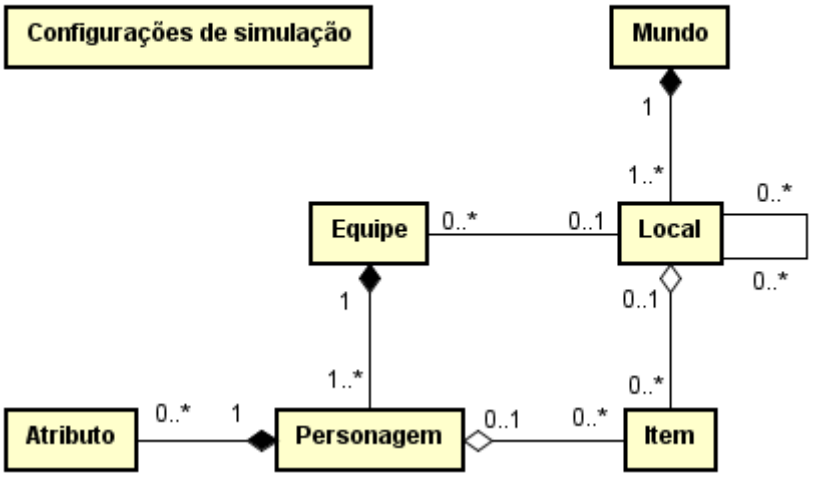

Figura 33: Diagrama de classes para os artefatos de entrada

O mundo de jogo pode representar o jogo inteiro, um level, ou uma parte de level que o *designer* queira construir. O mundo pode ser composto de quantos locais forem necessários para que a ideia na mente do *designer* seja transportada para o ambiente de simulação com a precisão desejada.

Os locais possuem conhecimento dos seus adjacentes e podem abrigar equipes, itens ou até mesmo não conter nada, funcionando apenas como um local de passagem.

Cada personagem pode possuir atributos e itens. Os atributos representam vários tipos de recursos que podem ser usados para controlar vida, ataque, velocidade e outras particularidades que o *designer* queira incluir nos personagens. Os itens podem ser utilizados para alterar de alguma forma o valor desses recursos. Esse modelo é abrangente o suficiente para permitir ao *designer* a construção de uma grande variedade de personagens.

5.3.2 Simulador de jogo

O simulador de jogo deve cumprir certos requisitos para assegurar o funcionamento adequado do ambiente proposto. O primeiro passo, ao receber os artefatos de entrada, é guardar o estado inicial para que os artefatos possam ser restaurados antes do início de cada simulação. Logo após, a equipe de heróis é colocada no primeiro local. Em cada local a sequência de eventos ocorre em três etapas:

i. *Pré-conflito*: Preparação para o conflito. Os personagens podem utilizar itens para melhorar ou restaurar atributos;

ii. *Conflito*: Resolução das ameaças presentes no local, como batalhas;

iii. *Pós-conflito*: Coleta de itens e recompensas, caso existam.

Em seguida, a equipe de heróis se desloca para um dos locais adjacentes e a sequência descrita ocorre novamente. Esse processo se repete até que ocorra uma das condições de paradas:

- A equipe de heróis atingiu o local final; ou
- Todos os heróis estão mortos.

Em ambos os casos a instância de simulação termina.

Durante uma simulação, o simulador deve gravar os dados de cada evento que ocorrer e, ao final, sumarizar todos eles. Imediatamente, os artefatos são restaurados às condições iniciais e a próxima simulação tem início. Após a ocorrência da última simulação o simulador deve consolidar os resultados de todas as anteriores e fornecer os dados armazenados. A Figura 34 apresenta o diagrama de estados do processo descrito.

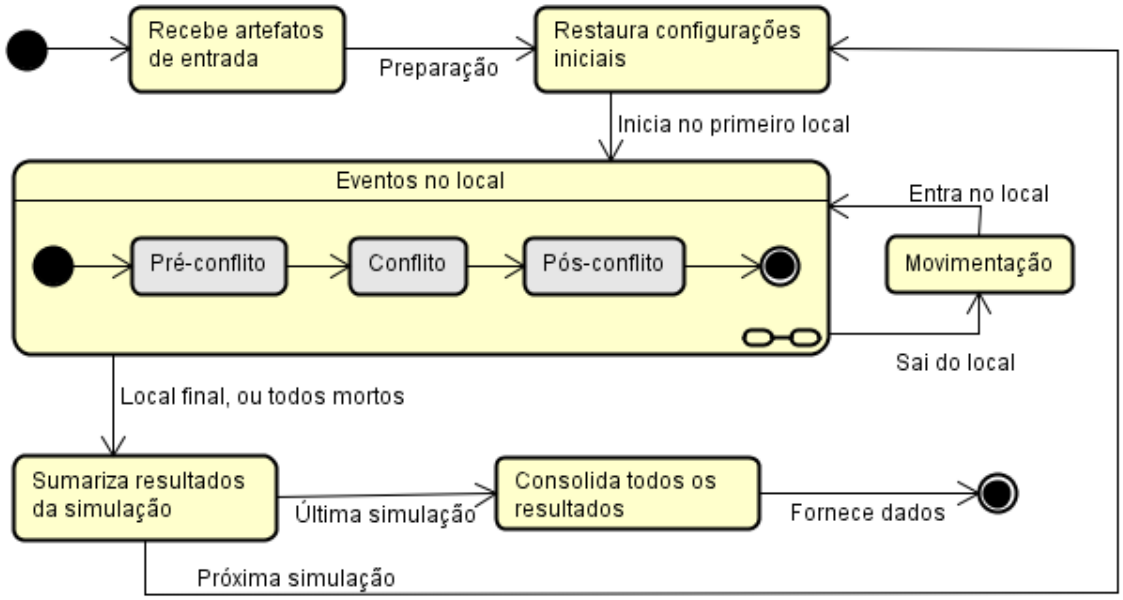

Figura 34: Diagrama de estados com requisitos do simulador

Os requisitos do simulador permitem a simulação de qualquer jogo ou parte de jogo que possui as mecânicas de adentrar em um local, resolver conflitos, coletar itens e avançar.

5.3.3 Artefatos de saída

Todos os eventos que ocorrem são dados de simulação e devem ser armazenados separadamente, com a finalidade de permitir a análise de uma simulação específica ou do conjunto de todas. Em cada simulação uma série de eventos podem ocorrer. A seguir é apresentada uma lista de possíveis eventos, mas que não os limita:

- Entrada em local;
- Saída de local;
- Item coletado em local;
- Item utilizado;
- Batalha entre equipes: Quantidade de turnos; personagem ataca outro personagem; personagem defende ou evita; personagem é atingido; personagem morre.

A partir das informações armazenadas podem ser gerados relatórios e gráficos demonstrando, por exemplo, a porcentagem de sucesso dos heróis em um conjunto de simulações, a quantidade média ou variação de algum atributo específico dos personagens ao longo das simulações, dados estatísticos de batalhas, log detalhado com sequência de eventos, etc.

As informações geradas constituem a base para a *Análise* do *designer* avaliar se o comportamento do jogo está de acordo com o que foi imaginado e, então, tomar decisões de *Refinamento* visando aprimoramento do jogo e balanceamento de atributos.

5.4 Contribuições em *playability*

O ambiente de simulação proposto fornece mecanismos que podem favorecer o balanceamento dos elementos que compõem o jogo. Tais mecanismos contribuem para o aprimoramento dos critérios *desafio* e *habilidade do jogador* do modelo de (SWEETSER & WYETH, 2005). Com relação ao modelo de (SÁNCHEZ *et al.*, 2012), a Figura 35 destaca algumas propriedades.

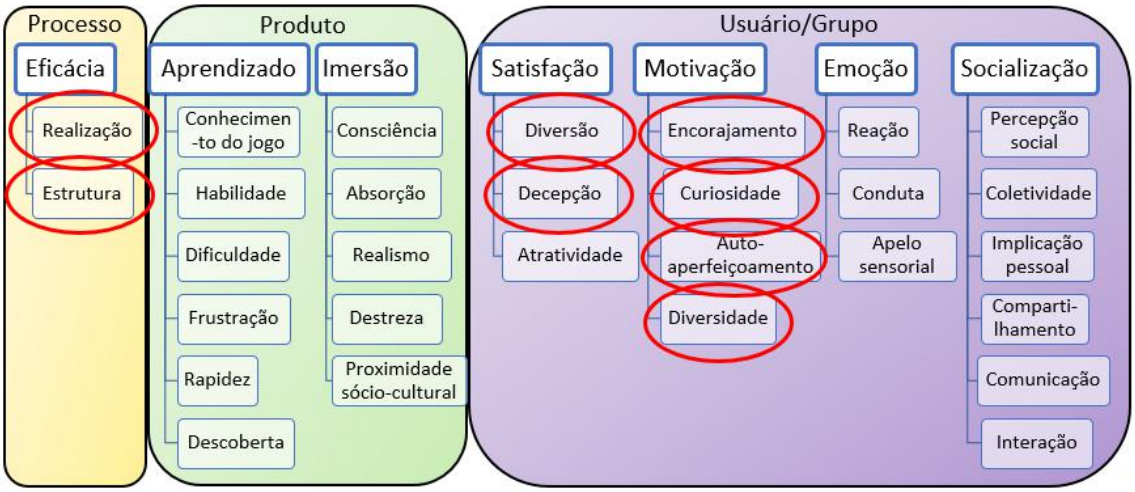

Figura 35: Atributos e propriedades do modelo de *playability*. Adaptado de (SÁNCHEZ *et al.*, 2012)

A proposta de ambiente de simulação suporta as propriedades em destaque, de forma que o *designer* pode utilizar o *ciclo de testes* para aprimorá-las. A seguir são apresentados detalhes sobre os atributos e propriedades que interessam a esse trabalho.

5.4.1 Eficácia

O atributo Eficácia reflete os recursos necessários para oferecer aos jogadores experiência de entretenimento, enquanto atingem os objetivos do jogo e chegam ao seu final. Pode ser analisado como o uso correto dos desafios no jogo, a estrutura dos objetivos e adaptação ao controle das ações no jogo. Eficácia possui as seguintes propriedades:

- Realização: existem jogadores que querem atingir 100% de um jogo, realizando todas os objetivos opcionais e existem jogadores que se interessam somente pelos objetivos principais. A Realização é refletida principalmente no número de elementos opcionais que foram atingidos num jogo, mas deve ser medida de acordo com o grau de interesse do jogador durante sua busca pelos objetivos, de acordo com seu perfil.
- Estrutura: um jogo é bem estruturado quando apresenta um bom balanceamento entre os diversos objetivos a serem alcançados e os desafios a superar, de modo que os jogadores permaneçam envolvidos e se divirtam enquanto estão jogando. Essa propriedade também oferece informação para

readaptar a experiência de jogo aos diferentes perfis de jogador. Por exemplo, o número de desafios (ou inimigos) numa área pode ser maior para um jogador avançado do que para um jogador casual.

5.4.2 Satisfação

É um atributo muito subjetivo e difícil de medir, pois depende da preferência de cada jogador. Aspectos que lhe despertam mais atenção podem contribuir para a satisfação, como personagens bem construídos, mecânicas simples ou complexas, gráficos, história, desafios etc. A estrutura do jogo exerce um forte impacto na satisfação. Duas características da satisfação são apoiadas pela proposta de ambiente de simulação:

- Diversão: é o quanto de diversão o jogador sente ao jogar o jogo. Se o jogo não for divertido os jogadores nunca se sentirão satisfeitos.
- Decepção: o grau de decepção é uma propriedade negativa e conectada à personalidade do jogador. Um jogo em que o jogador perde muitas vezes pode causar frustação a ponto de abandonar o jogo.

5.4.3 Motivação

A motivação pode ser definida como um conjunto de características que exigem que o jogador se mantenha realizando ações específicas até que elas sejam concluídas. Jogos com alto grau de motivação oferecem elementos que fazem os jogadores persistirem para superar os desafios. A motivação pode ter as propriedades a seguir:

- Encorajamento: é a confiança do jogador ao enfrentar novos desafios. Para aumentar a motivação o jogo deve minimizar a frustração, oferecendo recompensas adequadas de acordo com o grau de exigência do desafio. Um desafio quase impossível de ser vencido também pode fazer com que os jogadores percam o interesse no jogo.
- Curiosidade: estímulo da descoberta do que vem pela frente no jogo. Pode ser estimulada pela inclusão de elementos opcionais, objetivos e desafios que oferecem ao jogador liberdade para interagir com diversos elementos.
- Auto-aperfeiçoamento: o personagem ou elemento de jogo controlado pelo jogador é capaz de desenvolver novas habilidades ou se tornar mais poderoso conforme avança no jogo, ou se atinge determinados objetivos.

- Diversidade: a diversidade dos elementos de jogo torna o jogo mais atrativo para os jogadores e reduz a probabilidade de monotonia. A inclusão, por exemplo, de elementos de sorte e aleatoriedade pode conferir resultados agradáveis aos jogadores, de forma que seja quase impossível um jogo exatamente igual ao anterior.

6. Pegasus: Simulador de Jogos de Progressão

Este capítulo introduz a ferramenta Pegasus de simulação de jogos de progressão. O nome da ferramenta é um acrônimo gerado a partir da combinação das palavras *Progress Game Simulator*. A ferramenta, construída para avaliar a proposta apresentada no capítulo anterior, foi desenvolvida na linguagem de programação *Python*, de acordo com práticas de padrões de projeto de *software* (GAMMA *et al.*, 1995).

As seções desse capítulo apresentam o simulador Pegasus, detalhes da arquitetura, artefatos e como eles refletem os padrões de *design* de jogos, os componentes auxiliares, e os padrões de projeto de *software* utilizados.

6.1 Arquitetura

A arquitetura do simulador foi concebida visando a modularidade do sistema. Dessa forma, cada parte ou camada do sistema pode ser evoluída de forma independente sem afetar as demais. As camadas são responsáveis, de forma independente, pelo motor de simulação, armazenamento de resultado das simulações, processamento do resultado para geração dos artefatos de saída.

Uma outra camada é responsável pela comunicação entre o simulador e o usuário, e entre as demais camadas. Ou seja, cada uma das camadas independentes preocupa-se apenas com suas atividades específicas, interagindo apenas com a essa camada que realiza a comunicação.

A Figura 36 revela o conteúdo do simulador.

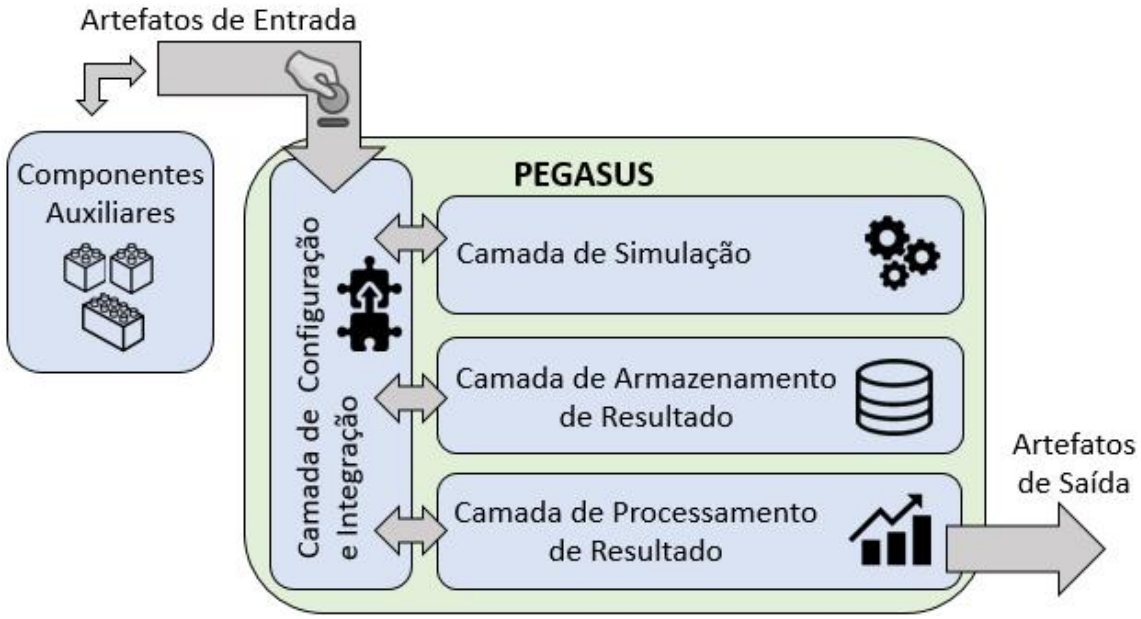

Figura 36: Arquitetura da ferramenta Pegasus

A partir da ideia descrita, o simulador construído é composto de quatro camadas. São elas:

- Camada de Configuração e Integração;
- Camada de Simulação;
- Camada de Armazenamento de Resultado; e
- Camada de Processamento de Resultado.

A Camada de Configuração e Integração é responsável por receber os artefatos de entrada, abrigar as configurações de simulação e realizar a comunicação entre as demais camadas. O Capítulo 7 é dedicado a esta camada.

Os Componentes Auxiliares e a Camada de Configuração e Integração são compostos por diversos módulos, por esse motivo de encontram representados na Figura 36 com peças de *Lego*. Assim sendo, novos módulos podem ser construídos e integrados facilmente ao simulador, evidenciando sua característica de expansividade.

A Camada de Simulação é o núcleo do simulador, área em que ocorre o processamento de cada instância de simulação. A Camada de Armazenamento de Resultado trabalha concomitantemente, armazenando cada resultado de instância de simulação. A Camada de Processamento de Resultado é responsável por gerar os artefatos de saída a partir dos dados das simulações. O Capítulo 8 é dedicado a esta camada.

Os Componentes Auxiliares não fazem parte do simulador, portanto não são incluídos em uma camada. Tais componentes são módulos não-essenciais desenvolvidos para serem facilitadores dos *designers* para a manipulação do simulador e construção dos artefatos de entrada. Mais detalhes sobre o funcionamento e utilidade de cada um dos componentes auxiliares são fornecidos em seção posterior.

6.1.1 Camada de Simulação

O funcionamento e a integração da Camada de Simulação com as demais estão representados no diagrama de sequência da Figura 37. Como pode ser notado, o diagrama implementa os requisitos do simulador apresentados em 5.3.2. A camada de simulação utiliza os artefatos de entrada para executar determinado número de simulações.

Cada vez que um local é visitado os dados de eventos são passados para a Camada de Configuração e Integração, que por sua vez envia para a Camada de Armazenamento de Resultado. Quando uma simulação é finalizada também são passados outros dados de eventos.

Em cada simulação é executada continuamente a sequência *Pré-conflito, Conflito, Pós-conflito*, até que se alcance um local final ou que não haja mais nenhum herói vivo.

Na fase *Pré-conflito* é verificado se existem no local equipes vivas para ocorrer batalhas. Itens são utilizados de acordo com a estratégia configurada. Na fase *Conflito* ocorrem as batalhas de acordo com a mecânica de batalha escolhida. Todas as ações são posteriormente passadas para outra camada e armazenadas. Na fase *Pós-conflito* são coletados os itens do local e, logo após, a equipe de heróis se move para outro local de acordo com a estratégia de movimentação.

Depois da execução de todas as simulações a Camada de Configuração e Integração disponibiliza todos os resultados armazenados para a Camada de Processamento de Resultado. Esta produz os artefatos de saída conforme configurações do usuário.

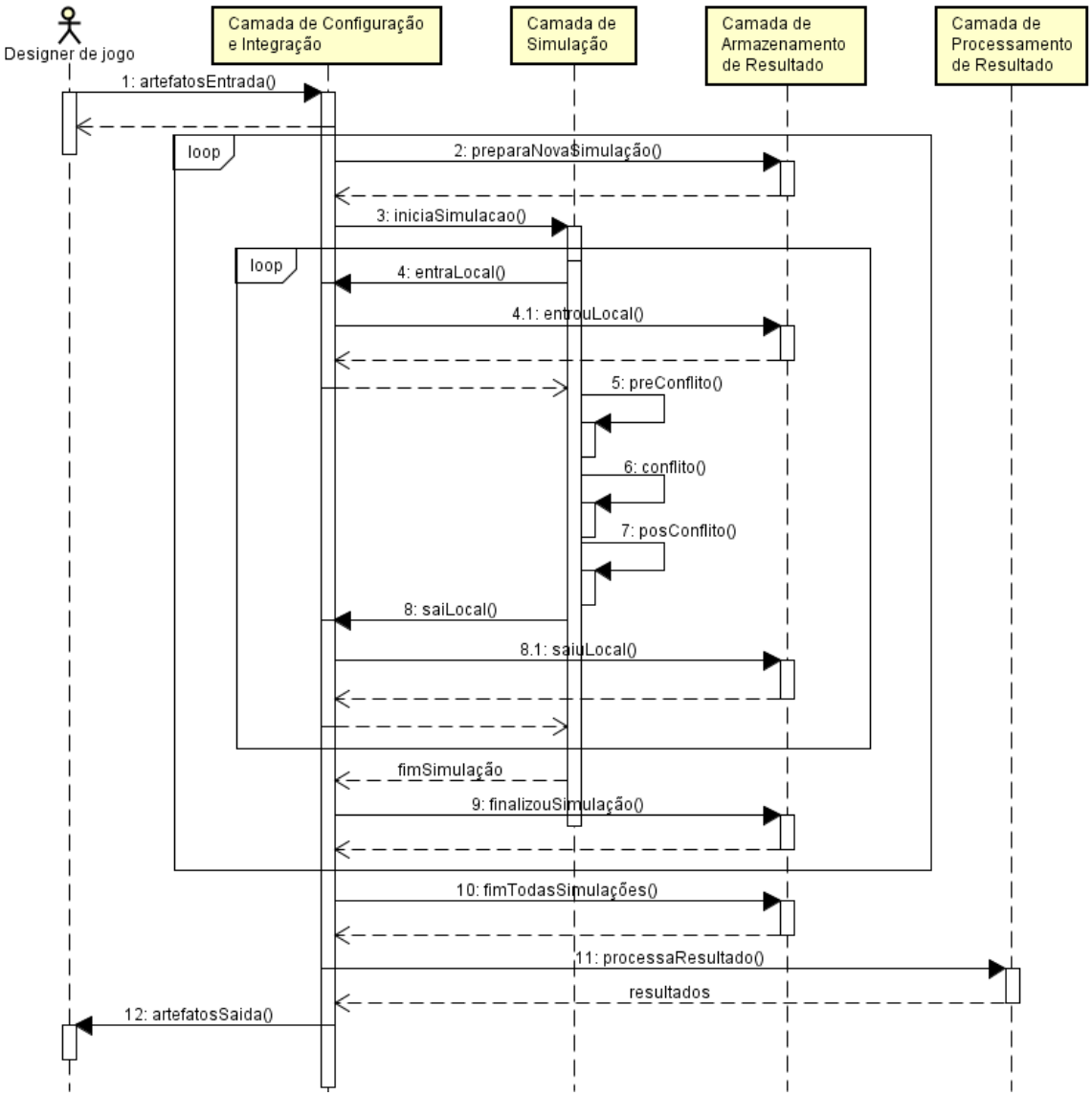

Figura 37: Diagrama de sequência do simulador

6.1.2 Camada de Armazenamento de Resultado

Essa camada é responsável por armazenar os dados de todos os acontecimentos de simulação. Cada grupo ou bateria de simulação é composta por várias instâncias de simulação e contém os artefatos de entrada com seus valores iniciais. Em cada simulação acontecem visitas aos diversos locais, em uma determinada ordem, onde podem haver itens e batalhas com inimigos. As batalhas são sequências de eventos de ataque e defesa entre os envolvidos e só terminam quando uma das equipes derrota todos os seus adversários presentes no local.

A Figura 38 apresenta um diagrama com o esquema de classes para armazenamento de resultados.

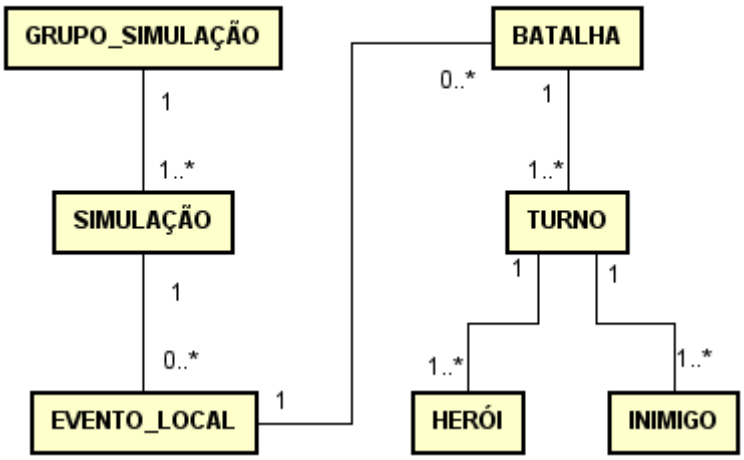

Figura 38: Classes de armazenamento de resultados

6.2 Artefatos de entrada

A presente seção apresenta cada um dos artefatos de entrada e como eles refletem os padrões de *design* de jogos. Eventualmente são apresentados diagramas de classes contendo representações simplificadas entre as principais classes e como estas se relacionam. Os *designs* de jogo mencionados nas próximas subseções aparecem destacados em itálico para facilitar sua identificação.

6.2.1 Personagem, Equipe, Atributo

Os personagens são os elementos que transitam pelo *mundo de jogo*. Cada personagem pode ser modelado refletindo um *avatar*, porém evidentemente o usuário não possui controle sobre este, afinal não se trata de um jogo, mas uma simulação.

Cada personagem pode ter nome, vida, um conjunto de atributos e um conjunto de itens no inventário. Dentre as principais ações se destacam atacar, defender, usar item, ser atingido. A vida foi mantida como atributo de personagem e não dentro do conjunto de atributos, em companhia dos outros atributos. Isso ocorre devido a obrigatoriedade de o personagem possuir o atributo vida para se manter ativo na simulação. Todos os demais atributos não são obrigatórios e o *designer* cria de acordo com sua intenção. A Figura 39 apresenta um diagrama de classes para personagem, equipe, atributo e item.

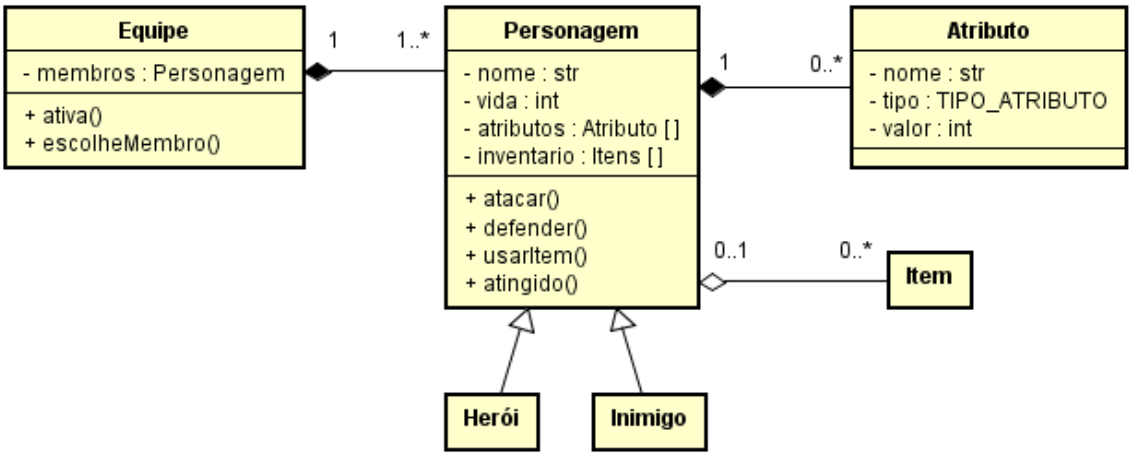

Figura 39: Classe Personagem e outras

O nome do personagem serve apenas para identificá-lo dentre os demais personagens de uma simulação. A vida é um valor que pode ser decrementado até chegar a zero. Quando isto ocorre o personagem se torna inativo.

Os atributos são os *recursos* que permitem construir inúmeros personagens diferentes, com suas variações e combinações. Para os propósitos de simulação há três tipos possíveis: ataque, defesa, agilidade. Não é obrigatório que um personagem possua algum destes, mas pode inclusive possuir mais de um de cada tipo. As intensidades das ações de um personagem são incrementadas de acordo com os valores dos atributos que este possui, conforme especificado na Tabela 2.

Tabela 2: Tipos de atributo

Tipo de atributo	**Descrição**
Ataque	Influencia no dano que o personagem pode causar.
Defesa	Influencia na suavização do dano que o personagem pode receber.
Agilidade	Influencia na velocidade para realizar o próximo ataque.

O inventário é o conjunto de *itens* que o personagem pode carregar. Mais detalhes sobre *itens* são fornecidos em seção posterior.

O personagem deve fazer parte de uma equipe, que por sua vez funciona como o padrão de *design* de jogo *unidade*. Enquanto houver pelo menos um personagem ativo a

equipe é considerada ativa. À equipe podem ser aplicadas diferentes opções de escolha de personagem para realização de determinadas ações. Isso ocorre de acordo com a estratégia utilizada pelo usuário.

Há uma relação de composição entre equipe e personagem, e entre personagem e atributo. Entre personagem e item a relação é agregação. Personagem pode ser especializado em herói ou inimigo. Estruturalmente ambos os tipos são iguais, suas diferenças são nítidas apenas a nível de código.

Um *chefão* pode ser construído como um *inimigo*, porém com valores de seus atributos mais elevados, conforme o padrão de *design chefão* atesta. Por esse motivo não existe uma classe diferenciada para o *chefão*.

O *designer* de jogo, usuário da ferramenta, tem suporte para a criar personagens por meio do componente auxiliar *fábrica de personagens*, desenvolvido com a finalidade de facilitar esse processo.

6.2.2 Local, Mundo

Os locais são os elementos que demarcam espaços por onde transitam as equipes. Um local sempre contém um número único para identificá-lo dos demais locais do *mundo de jogo*. Quando há alguma equipe inimiga no local, a equipe de heróis precisa derrotá-la para poder continuar sua jornada.

São opcionais ao local:

- *Cadeado*: O local só pode ser acessado se a equipe visitante possui a *chave* correspondente. O padrão de *design área inacessível* acontece quando o *cadeado* é colocado sobre uma área opcional, isto é, uma área dispensável de transitar para chegar ao local final da simulação. Quando o *cadeado* se localiza em um local imprescindível para se chegar no final da simulação, trata-se do padrão o*bstáculo*.
- *Equipe*: Se existir alguma, derrotá-la é o objetivo no local e a primeira ação a ser realizada é combatê-la. Somente após a *eliminação* da equipe residente é possível obter a *recompensa* e progredir.
- *Itens*: Podem existir em locais com ou sem equipe de inimigos. Quando o local possui equipe de inimigos os itens caracterizam a *recompensa* da batalha. Quando não há inimigos os itens são coletados no momento em que a equipe de herói adentra o local. Nesse caso os itens são *pick-ups*, isto é, coletáveis.

Entre mundo e local a relação é composição. Entre local e item há agregação. A Figura 40 apresenta a relação entre essas classes.

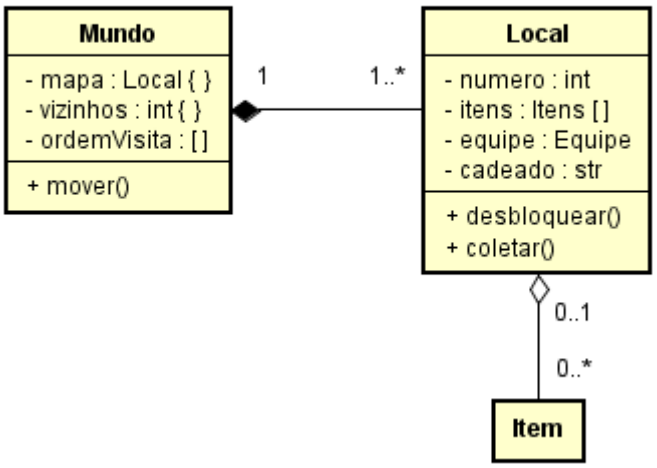

Figura 40: Mundo, Local e Item

O mundo é o maior ambiente da simulação, composto por um mapa com a relação de todos os locais e as ligações existentes entre seus vizinhos. Um mundo pode representar tanto o *mundo de jogo* completo, como também pode ser a representação de um *level* que o *designer* queira testar e refinar. O *objetivo* sempre é conseguir alcançar o local final.

A ação de mover é executada para a movimentação de um local a outro. A escolha do próximo *local* a ser visitado depende da possibilidade de ser acessado e da estratégia de movimentação configurada. O simulador permite integrar novas estratégias que venham a ser desenvolvidas.

6.2.3 Item

Itens são elementos inicialmente dispostos nos locais, seguindo o conceito do padrão *pick-up* de *design* de elemento de jogo. Seu atributo nome unicamente o identifica; a quantidade informa por quantas vezes o item pode ser utilizado; e o tipo especifica a natureza do item. São cinco diferentes possibilidades, conforme consta na Tabela 3.

Tabela 3: Tipos de item

Tipo de item	**Descrição**
Chave	Essencial para acessar local protegido por *cadeado.*

Vida/Ataque/Defesa/Agilidade	Permite incrementar em certo valor o atributo vida do personagem ou um dos demais atributos.

A ação de usar um item basicamente consome uma porção do item, quando possível, e atualiza sua quantidade. A organização de item e suas especializações podem ser observadas na Figura 41.

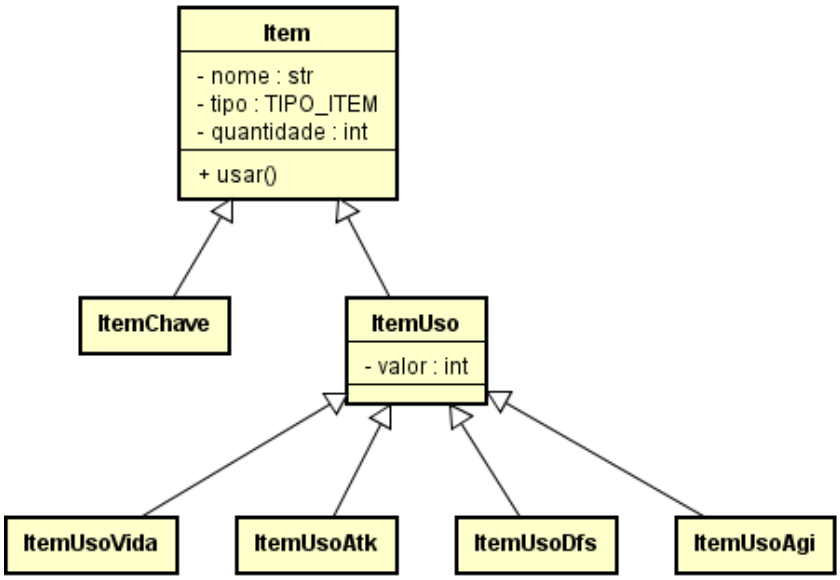

Figura 41: Item e subclasses

Os quatro *itens* que aparecem na parte de baixo da Figura 41 possuem o atributo valor, e são inspirados no padrão *power-up* de *design* de jogo. Isto é, quando utilizados aperfeiçoam determinados *recursos*.

6.3 Componentes Auxiliares

A presente seção detalha componentes não-essenciais desenvolvidos para o simulador, seu funcionamento e capacidade de expansão. Os componentes auxiliares são utilizados para dar suporte à criação de artefatos de entrada.

6.3.1 Fábrica de personagens e Fábrica de itens

Constitui-se como facilitador para criar personagens e itens de forma simples. No caso de personagens o tipo, nome e atributos do herói ou inimigo são informados como

valores. No caso de itens, são informados o tipo e demais atributos. A fábrica cria e retorna o artefato especializado conforme os dados fornecidos.

As construções tanto da fábrica de personagens quanto da fábrica de itens seguem os padrões de projeto *Abstract Factory* e *Factory Method*. A Figura 42 exibe o posicionamento da fábrica de personagens. A fábrica de itens é implementada de maneira análoga.

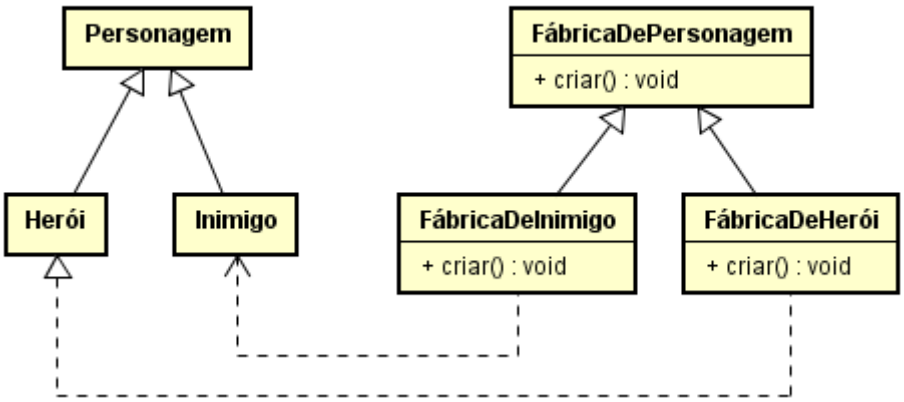

Figura 42: Fábrica de personagens

6.3.2 Formato e leitor de XML para Mundo, Local, Inimigo e Item

Componente para ler arquivo XML em formato específico com dados do mundo, local, inimigo, item e como estão dispostos visando agilidade na construção dos mencionados artefatos de entrada.

O arquivo XML deve iniciar com um elemento raiz de qualquer nome, seguido por um ou mais elementos *world*. Este por sua vez pode conter um ou mais elementos *place*, que podem conter um ou mais elementos *foe* e *item*.

Os elementos raiz e *world* não possuem nenhum atributo. Todos os atributos dos elementos *place, foe* e *item* são do tipo *string* e se encontram na Tabela 4, Tabela 5 e Tabela 6. Os atributos marcados com * são obrigatórios.

Tabela 4: XML – Elemento *place*

Atributo	**Descrição**
id *	Número do local.

neighbors	Números dos locais vizinhos possíveis de mover ao sair do presente local. Os números são separados por vírgula.
idLock	Define o *cadeado* conforme exposto em 6.2.2.
final	Recebe o valor *true* para indicar que o local marca o final da simulação.

Tabela 5: XML - Elemento *foe*

Atributo	**Descrição**
name *	Nome do inimigo.
lif	Valor do atributo vida do inimigo.
atk	Valor do atributo de ataque do inimigo.
dfs	Valor do atributo de defesa do inimigo.
agi	Valor do atributo de agilidade do inimigo.
quantity	Quantidade de inimigos.

Tabela 6: XML - Elemento *item*

Atributo	**Descrição**
id *	Nome do *item.*
type	Tipo do *item*. Possibilidades: *key*, *life*, *atk*, *dfs*, *agi*.
quantity	Quantidade do *item*.
value	Valor do *item.*

Os elementos *item* e *foe* são opcionais ao XML. Deve haver um ou vários elementos *place*. É possível utilizar mais de um elemento *world*, porém o componente cria os artefatos em apenas uma instância de mundo. Com efeito, utilizar mais de um elemento *world* no XML serve meramente para facilitar a organização ou visualização do *mundo de jogo* modelado. A Figura 43 exemplifica esse fato.

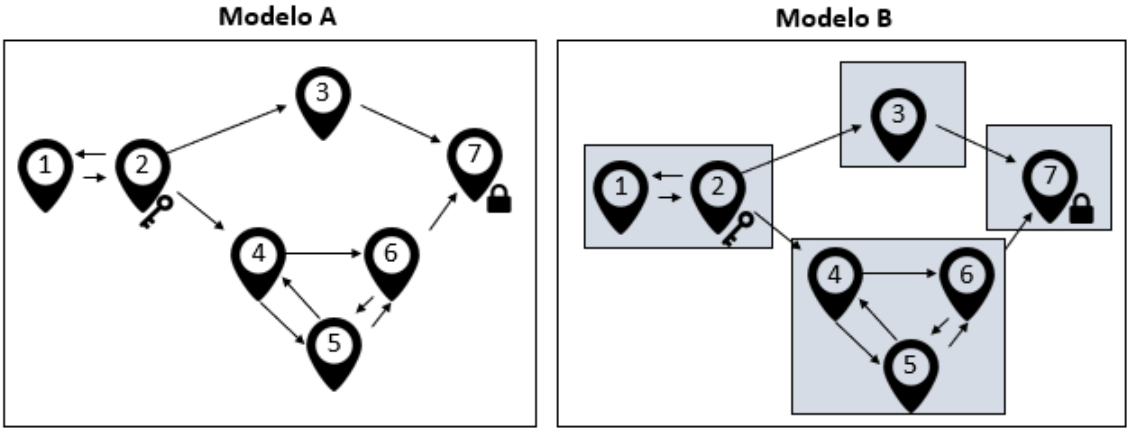

Figura 43: Modelos equivalentes

Em ambos os modelos da Figura 43 há um *mundo de jogo* composto por sete locais. Embora no modelo B o *mundo de jogo* seja visualizado como se fosse uma composição de quatro agrupamento de locais, ou quatro submundos, tal construção não difere do modelo A no que diz respeito à simulação. Tal possibilidade de construção pode ajudar o *designer* a pensar em um local constituído de vários outros locais, o que está de acordo com padrão *level* de *design* de elemento de jogo.

O XML a seguir retrata o modelo A.

```
<root>
  <world>
    <place id='1' neighbors='2'/>
    <place id='2' neighbors='1,3,4'>
      <item id='silverKey' type='key'/>
    </place>
    <place id='3' neighbors='7'>
      <foe name='Troll' lif='80' atk='14' dfs='8' agi='4'/>
    </place>
    <place id='4' neighbors='5,6'/>
    <place id='5' neighbors='4,6'>
      <item id='potion' type='lif' value='50' quantity='2'/>
    </place>
    <place id='6' neighbors='5,7'/>
    <place id='7' idLock='silverKey' final='true'/>
  </world>
</root>
```

E o XML do modelo B seria:

```
<root>
  <world>
    <place id='1' neighbors='2'/>
    <place id='2' neighbors='1,3,4'>
      <item id='silverKey' type='key'/>
```

```
    </place>
  </world>
  <world>
    <place id='3' neighbors='7'>
      <foe name='Troll' lif='80' atk='14' dfs='8' agi='4'/>
    </place>
  </world>
  <world>
    <place id='4' neighbors='5,6'/>
    <place id='5' neighbors='4,6'>
      <item id='potion' type='lif' value='50' quantity='2'/>
    </place>
    <place id='6' neighbors='5,7'/>
  </world>
  <world>
    <place id='7' idLock='silverKey' final='true'/>
  </world>
</root>
```

Vale destacar novamente que ambos os modelos são equivalentes e, ao serem fornecidos para o componente leitor de XML, produzem os mesmos artefatos.

6.4 Padrões de projeto utilizados

Os padrões de projeto de *software* são modelos e práticas formalizadas de resoluções de problemas comuns que o programador pode se deparar durante a construção de um sistema de informação. Os padrões de projeto ganharam notoriedade com a publicação do livro (GAMMA *et al.*, 1995), também conhecido como livro da “gangue dos quatro”, tradução livre de *Gang of Four*, alcunha pela qual seus autores são conhecidos.

Os seguintes padrões de projeto de *software* listados a seguir foram intencionalmente utilizados na construção da ferramenta de simulação:

- *Abstract Factory*: Utilizado na criação de personagens, itens e atributos.
- *Factory Method*: Utilizado na criação de personagens e itens.
- *Command*: Utilizado no personagem para recuperar diferentes atributos e itens.
- *Mediator*: Utilizado pelo sistema de batalhas para a interação dos personagens.
- *Strategy*: Utilizado na movimentação entre locais do mundo de jogo, na escolha de personagem para alguma ação ou ataque, na utilização do sistema de batalha configurado.

- *Chain of Responsability*: Utilizado no sistema de batalhas de *Active Time Battle* (ATB) para invocar o próximo personagem a realizar a ação a cada momento, até o fim da batalha.
- *Singleton*: Praticamente todas as classes que representam elementos de jogo são *singletons*, como personagem, local, mundo, item, equipe. Este padrão também é utilizado no gerenciador de resultado, responsável pelo armazenamento dos dados das simulações.
- *Facade*: Utilizado no sistema de batalhas para facilitar o uso pela Camada de Simulação, e utilizado também na Camada de Processamento de Resultado para simplificar as opções de plotagem de gráficos.

7. Camada de Configuração e Integração

A Camada de Configuração e Integração possui duas funções principais. Uma delas é a integração, já mencionada no capítulo anterior. Essa camada funciona como a ponte de comunicação e passagem de dados entre as demais camadas do simulador.

A outra função é a de configuração. Os módulos dessa camada permitem que o *designer* escolha opções de configuração de simulação que podem produzir diferentes resultados. Essas configurações são conjunto de opções para guiar o comportamento do simulador em relação aos demais artefatos expostos nas seções anteriores.

Tais como os componentes auxiliares, os módulos podem ser expandidos para ganhar novas opções. Também há a possibilidade de criação de novos módulos. Os módulos se diferem dos componentes auxiliares por serem parte interna e essencial do simulador, enquanto os componentes auxiliares são partes externas que apenas facilitam a criação de artefatos.

A seguir a lista das possíveis configurações:

- Estratégia de escolha de personagem para atacar em batalha;
- Estratégia de escolha de personagem para se beneficiar de item;
- Estratégia de movimentação;
- Escolha de mecânica de batalha;
- Escolha da intensidade do gerador de aleatoriedade;
- Escolha das opções de resultado.

As próximas seções detalham cada módulo desenvolvido.

7.1 Estratégia de escolha de personagem

Módulo para decidir dinamicamente, em tempo de simulação, qual dos personagens integrantes de uma equipe irá sofrer ou realizar determinada ação.

As opções de estratégias implementadas se integram ao simulador segundo o padrão de projeto *Strategy*. A Tabela 7 apresenta tais opções.

Tabela 7: Estratégias de escolha de personagem

Estratégia	Descrição
Escolhe o primeiro	Escolhe o primeiro personagem colocado na equipe no momento de sua criação.
Escolhe aleatoriamente	Sorteia aleatoriamente um personagem dentre os integrantes da equipe.
Escolhe personagem com maior atributo X	Escolhe dentre a equipe o personagem que possui o maior valor de determinado atributo. Exemplo: O que possui maior valor de ataque.
Escolhe personagem com menor atributo X	Escolhe dentre a equipe o personagem que possui o menor valor de determinado atributo. Exemplo: O mais lento (possui menor valor de agilidade).

As estratégias de escolha de personagens são aplicadas em duas situações:

- Mecanismo de batalha: escolha do personagem da equipe adversária que será alvo do ataque;
- Coleta de itens: eleger o personagem da equipe que será beneficiado com algum item coletado.

7.2 Estratégia de movimentação por locais

Módulo para determinar o próximo local a ser visitado pela equipe. As opções de movimentação implementadas se integram ao simulador segundo o padrão de projeto *Strategy*, tal como as estratégias de escolha de personagem.

A Tabela 8 apresenta as opções implementadas. Um local está disponível quando é adjacente ao local atual e não possui cadeado, ou a equipe de heróis possui a chave correta do cadeado do local.

Tabela 8: Estratégias de movimentação

Estratégia	Descrição
Próximo	Move a equipe para o próximo local disponível, segundo seu número. Se nenhum dos próximos locais estiverem

	disponíveis, move a equipe para um local de número abaixo ou retorna ao local anterior.
Próximo aleatoriamente	Move a equipe para um local sorteado dentre todos os próximos disponíveis, segundo seu número. Se nenhum dos próximos locais estiverem disponíveis, move a equipe para um local sorteado dentre todos os disponíveis.
Mais longe	Move a equipe para o local mais afastado disponível, isto é, o local de maior número. Se não houver nenhum próximo local disponível, retorna ao anterior.
Novos locais	Move a equipe para um local sorteado dentre todos os não-visitados disponíveis. Se todos os disponíveis já foram visitados, move a equipe para o local mais afastado.

Novas estratégias podem ser construídas e integradas facilmente às existentes, tanto para movimentação quanto para escolha de personagem.

7.3 Configuração do sistema de batalha

As batalhas são conflitos que ocorrem nos locais entre as equipes presentes. O mecanismo de batalha possui diversos elementos de padrões de *design* de jogo, são eles: *combate, eliminação, objetivos, recompensas, aleatoriedade, jogos baseados em turno, jogos baseados em tempo*.

Os sistemas de batalhas constituem uma parte fundamental dos gêneros RPG e seus derivados. Ao longo dos anos muitos jogos surgiram com diferentes sistemas e derivados, dentre os quais se destacam os sistemas de batalhas por turnos e em tempo real (STENSTRÖM & BJÖRK, 2013), sendo este último mais recente.

Nessa tese foram construídas duas opções derivadas dos sistemas de batalhas por turnos: turnos alternados; e tempo ativo, tradução livre de *Active Time Battle* (ATB). O usuário pode escolher um destes para executar a simulação.

No sistema de turnos alternados todos os personagens realizam sua ação, um por vez. Após o último realizar sua ação inicia-se um novo ciclo, mantendo a mesma ordem de personagens. Essa ordem é definida pelo atributo *agilidade* dos personagens. Os personagens que não possuem esse atributo são os últimos a realizarem a ação. Dentre os

personagens que possuem o mesmo valor de *agilidade*, ou que não possuem esse atributo, a ordem é definida aleatoriamente.

O sistema ATB é popular em jogos clássicos de RPG desenvolvidos no Japão. A série de jogos *Final Fantasy* foi uma das pioneiras nesse sistema e introduziu inovações que mais tarde se tornariam comuns em jogos deste gênero (ANDITYA & SIHABUDDIN, 2016). Esse sistema introduz uma barra de tempo de ação que define o momento em que o personagem pode executar sua próxima ação na batalha. A *agilidade* do personagem determina a velocidade com que a barra é preenchida. A Figura 44 mostra uma batalha em *Final Fantasy VII* em que a barra de tempo de ação "*time*" está presente.

Figura 44: Batalha em *Final Fantasy VII*

Nos dois sistemas de batalha implementados a ação executada pelo personagem é o ataque a um dos adversários, condicionado pela estratégia escolhida. A força de ataque (FA) é calculada pela soma de todos os atributos de ataque (AA) do personagem, mais o valor da soma dos seus *itens* de ataque (IA) e bônus de ataque (BA) (1).

$$FA = \sum AA + \sum IA + BA \quad (1)$$

A força de defesa (FD) é calculada de maneira análoga, soma-se todos os atributos de defesa (AD) do personagem alvo do ataque, mais o valor da soma dos seus *itens* de defesa (ID) e bônus de defesa (BD) (2).

$$FD = \sum AD + \sum ID + BD \quad (2)$$

A diferença entre as forças de ataque e defesa constituem o dano sofrido, ou valor de ponto de vida perdido pelo personagem adversário que sofre o ataque (3).

$$D = FA - FD \quad (3)$$

Os bônus de ataque e defesa mencionados são valores aleatórios introduzidos para outorgar, em certa escala, característica de imprevisibilidade, usualmente encontrada em sistemas de batalhas.

7.4 Geração de aleatoriedade

O padrão *aleatoriedade* para *design* de jogo indica que a existência de imprevisibilidade nos jogos é algo desejável, por dificultar execuções idênticas do mesmo jogo. Esse padrão serve de auxílio para que um jogo possa ser reproduzido diversas vezes sem que se torne repetitivo ou monótono. Em inglês, existe o termo *replayability* para denotar essa propriedade.

O mecanismo de batalhas do simulador dispõe de valores bônus que podem ser somados ao ataque e/ou defesa durante o combate. O presente módulo é responsável por gerar números para bônus de maneira aleatória, assim como um rolar de dados ou giro de roleta. As proporções de valores máximo e mínimo do bônus em relação aos valores dos atributos de ataque e defesa são configurados separadamente.

Em jogos de RPG existe o termo *ataque crítico* para indicar um ataque perfeito, que geralmente causa o dobro de dano no adversário. Essa situação pode ser simulada quando ocorre aleatoriamente o valor máximo de bônus de ataque. Portanto, recomenda-se que o maior valor de bônus possa, no máximo, dobrar o valor original.

7.5 Configurações de saída

O próximo capítulo apresenta detalhes sobre os artefatos de saída. Esse módulo somente permite ao usuário escolher o diretório do computador em que os artefatos de saída serão gravados, permite ligar, ou desligar, as opções de produção de relatório resumido, detalhado, e geração de gráficos.

8. Camada de Processamento de Resultado

A Camada de Processamento de Resultado utiliza os dados gerados nas simulações para confeccionar os artefatos de saída.

Existem basicamente dois tipos de artefatos: relatórios e gráficos. Os relatórios podem ser resumidos ou detalhados e contém o registro das informações das simulações na ordem de ocorrência. Os gráficos procuram explicitar de maneira visual dados consolidados de todas as simulações. As próximas seções apresentam a descrição e exemplos dos artefatos.

8.1 Relatório

Os relatórios são como *logs* com os eventos das simulações gravados em arquivo texto.

A estrutura dos relatórios foi elaborada de forma a facilitar a leitura e coleta de dados do seu conteúdo. Dessa maneira incentiva-se a construção de novos componentes ou ferramentas para realizar análises computacionais do seu conteúdo e extração de novas informações.

8.1.1 Relatório resumido

As simulações aparecem no relatório numeradas em ordem crescente. Cada simulação é chamada no relatório de *Expedition*. O relatório contém marcações para indicar o início e o fim de uma simulação. Em cada simulação os dados de eventos registrados são:

- Entrada no local, indicando o número do local;
- Quantidade de turnos que durou a batalha no local, quando houver batalha.
- Nome e quantidade de cada item encontrado no local;
- Resultado final da simulação: vitória ou derrota da equipe de heróis;
- Total de locais visitados;
- Lista de todos os itens, na ordem em que foram coletados;
- Número do local onde a simulação terminou;

- Total de turnos de todas as batalhas;
- Média de turnos por batalha;
- Ordem de visita dos locais.

A Figura 45 mostra o trecho inicial de um relatório resumido, contendo informações da primeira simulação de um grupo de cinco mil simulações.

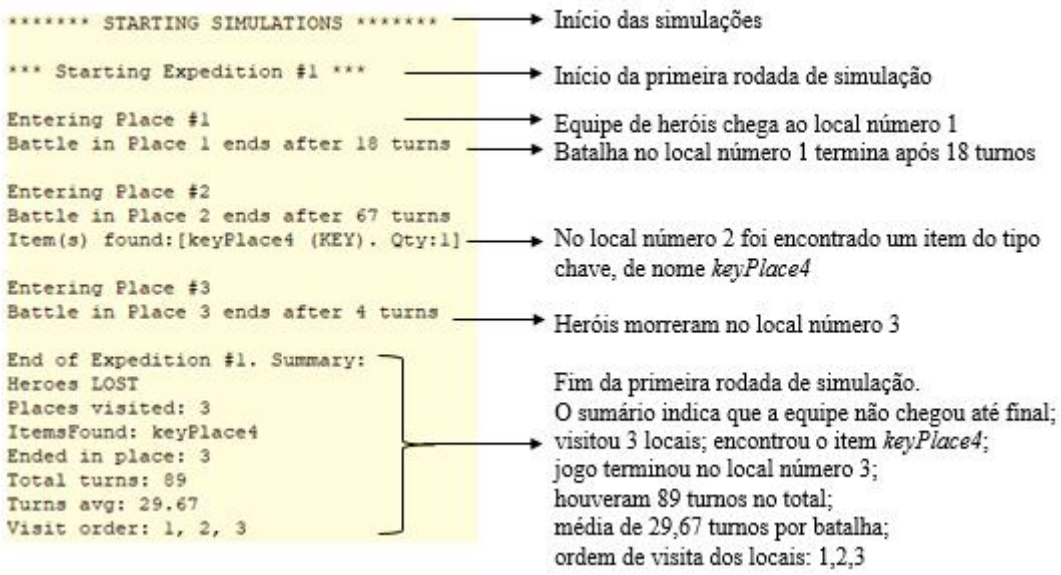

Figura 45: Trecho de relatório resumido – uma simulação

Como pode ser notado na figura acima, o relatório contém um sumário no final de cada simulação. No final do arquivo, após todas as simulações, o relatório também contém uma síntese de todas as simulações. Os dados presentes são:

- Número e porcentagem de vitórias da equipe de heróis;
- Número e porcentagem de derrotas da equipe de heróis;
- Média de locais visitados;
- Média de itens encontrados;
- Média do total de turnos por simulação;
- Média de turnos por batalha;
- Ordem mais comum de visita aos locais: quantidade e porcentagem.

A Figura 46 apresenta o exemplo do trecho final com a síntese de um relatório resumido após cinco mil simulações.

```
----------Simulation report after 5000 expeditions:----------
Heroes successes: 1600 (32.0%)
Heroes failures: 3400 (68.0%)
Avg visited places: 4.05
Avg items found: 1.49
Turns avg per expedition: 85.88
Turns avg per places: 24.5
Most common visit order: 1, 3, 5. Per 1250 times (25.0%)
```

Figura 46: Trecho de relatório resumido – síntese

O relatório da Figura 46 indica que a equipe chegou com sucesso ao final em 32% das vezes, falhando em outros 68%; teve uma média de 4,05 locais visitados; média de 1,49 itens encontrados; média de 85,88 turnos em cada simulação; média de 24,5 turnos por batalha; e a ordem mais comum de visita aos locais foi: 1,3,5, ocorrendo por 25% das vezes.

8.1.2 Relatório detalhado

O relatório detalhado contém todas as informações do relatório resumido, acrescido das seguintes informações:

- Nome e listagem dos valores dos atributos de cada herói e inimigo sempre que um local é adentrado.
- Descrição de cada turno de batalha: Informa personagem que realiza o ataque, o alvo, a força de ataque, força de defesa, valor do dano sofrido e novo valor de vida do alvo.

A Figura 47 contém um trecho de relatório detalhado. Para que a figura não ficasse muito extensa ela exibe apenas os três primeiros turnos dos vinte e nove que ocorreram.

```
Entering Place #3
Heroes data:                                                                    A
Name: Knight. Life: 71.0. Attributes: Sword (ATTACK): 10. Shield (DEFENSE): 8. Agi (AGILITY): 10.
Name: Wizard. Life: 25.0. Attributes: Wand (ATTACK): 12. Wand (DEFENSE): 6. Agi (AGILITY): 7.
Enemies data:                                                                   B
Name: Minion 1. Life: 20. Attributes: Atk (ATTACK): 5. Dfs (DEFENSE): 2. Agi (AGILITY): 10.
Name: Minion 3. Life: 20. Attributes: Atk (ATTACK): 5. Dfs (DEFENSE): 2. Agi (AGILITY): 10.
Name: Minion 3. Life: 20. Attributes: Atk (ATTACK): 5. Dfs (DEFENSE): 2. Agi (AGILITY): 10.
Name: Hunter. Life: 30. Attributes: Axe (ATTACK): 15. Shield (DEFENSE): 6. Agi (AGILITY): 7.
Hunter attacks Wizard. Atk pw: 20. Dfs pw: 7.6. Dam: 12.4. Wizard life: 12.6   C
Minion 1 attacks Wizard. Atk pw: 9. Dfs pw: 10.8. Dam: 0. Wizard life: 12.6
Knight attacks Minion 2. Atk pw: 16. Dfs pw: 3.6. Dam: 12.4. Minion 2 life: 7.6
.
.
.
Battle in Place 3 ends after 29 turns
No Items
```

Figura 47: Trecho de *log* detalhado

No exemplo da Figura 47 a equipe de heróis chega ao local número 3 e encontra uma equipe de inimigos. Os trechos identificados com A e B descrevem nomes e valores dos atributos dos personagens da equipe dos heróis e da equipe dos inimigos, respectivamente. O trecho identificado com C relata turno a turno o andamento da batalha.

O relatório detalhado também possui um sumário ao final de cada simulação e a síntese de todas as simulações ao final do relatório. Esses trechos são compostos das mesmas informações que o relatório resumido.

8.2 Gráficos

A Camada de Processamento de Resultado produz gráficos a partir dos dados reunidos pelas simulações com o intuito de simplificar a visualização de informações. As próximas subseções detalham os gráficos que podem ser gerados

8.2.1 Gráfico do local fim da simulação

Exibe a quantidade de vezes que a simulação foi encerrada em cada local. Pode ser apresentado em gráfico de barras ou em gráfico circular (formato de pizza). A figura a seguir apresenta um exemplo em que a simulação terminou 14,6% das vezes no local número 5, 14,9% no local número 6 e 70,5% no local número 7.

Esse gráfico pode indicar para o *designer* algum local com dificuldade elevada que faz com que os heróis morram ali com frequência. Também pode indicar com que frequência o local final é atingido.

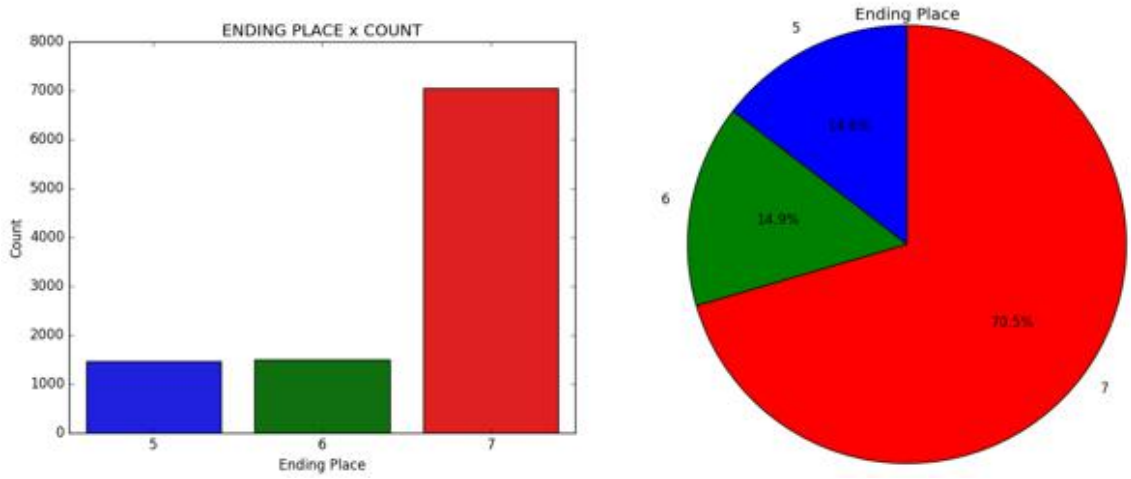

Figura 48: Gráficos local fim da simulação

8.2.2 Gráfico da variação de atributo por personagem

Exibe a variação de algum atributo de algum personagem em cada local, ao longo das simulações. Pode ser apresentado em gráfico de dispersão ou em diagrama de caixa (*box plot*). A Figura 49 e Figura 50 apresentam um exemplo da variação do atributo *vida* de um personagem de nome *Knight* em cada local, ao longo das simulações.

Analisar a variação dos atributos de força, ataque, defesa ou agilidade fazem sentido quando existe uma grande quantidade de itens que modificam tais atributos. A análise do atributo de vida, por personagem, pode auxiliar a identificar algum personagem que esteja muito mais forte ou fraco em relação aos demais.

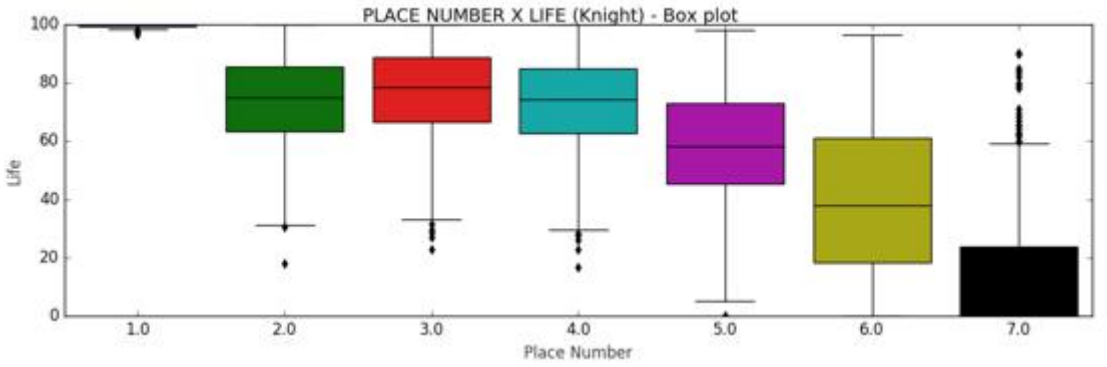

Figura 49: Gráfico variação de atributo por personagem – *box plot*

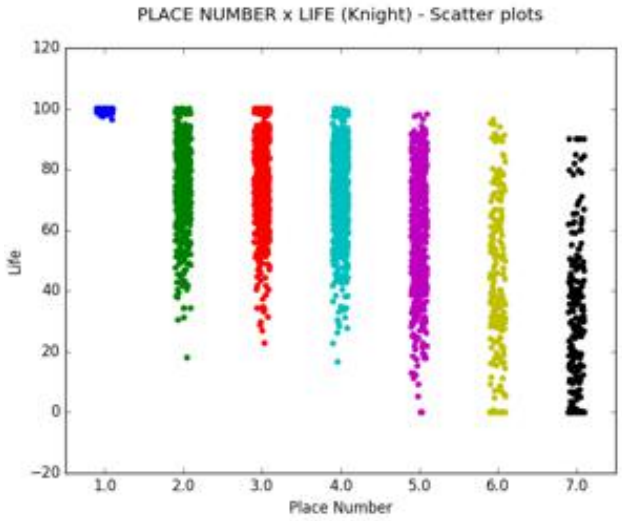

Figura 50: Gráfico variação de atributo por personagem – dispersão

8.2.3 Gráfico da média de atributo com todos personagens

Exibe o valor médio de algum atributo dos personagens, em cada local. Todos os personagens da equipe de heróis são apresentados juntos no mesmo gráfico de linha. A Figura 51 apresenta um exemplo com a média do atributo *vida* dos personagens *Knight*, *Amazon* e *Wizard*, em cada local.

Assim como o gráfico anterior, o atributo de vida é o que faz mais sentido de ser analisado pelos mesmos motivos. Esse gráfico pode ajudar a descobrir algum local em que os personagens percam mais vida do que se gostaria, sendo possivelmente um local com dificuldade elevada. Ou a situação inversa, algum local que o *designer* gostaria que fosse mais difícil não está apresentando tanta dificuldade.

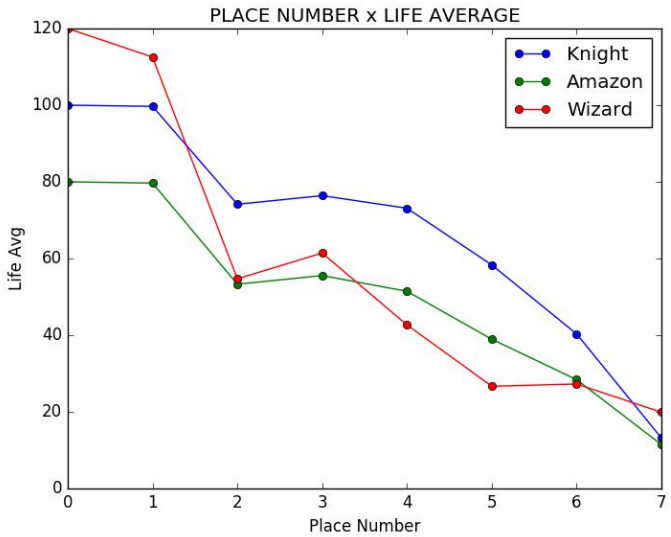

Figura 51: Gráfico média de atributo dos personagens

8.2.4 Gráfico de quantidade de turnos

Exibe a variação da quantidade de turnos de batalha em cada local, ao longo das simulações, em um gráfico de dispersão. A Figura 52 apresenta um exemplo de jogo com sete locais, onde em todos ocorreram batalhas.

Esse gráfico auxilia o *designer* a identificar a duração das batalhas em cada local. No exemplo a seguir, as batalhas no local 1 apresentam a menor duração, as batalhas nos locais 2 e 4 apresentam consistentemente uma duração maior. O local número 7 apresenta uma grande variação da duração de batalha. Tem batalhas com mais de 60 turnos e outras

com menos de 10, isso pode significar que a batalha termina precocemente devido à derrota da equipe de heróis.

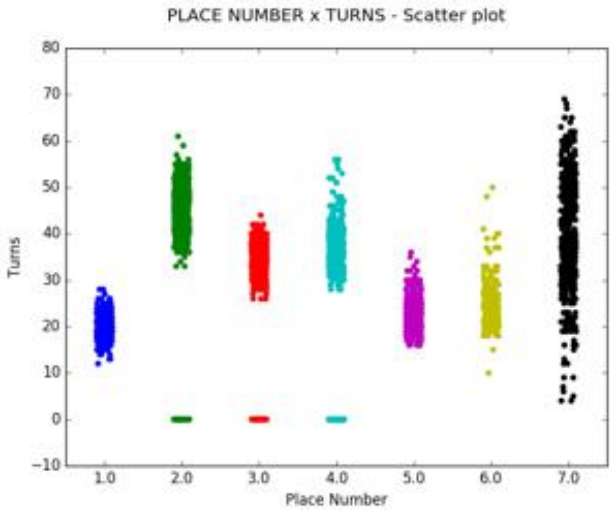

Figura 52: Gráfico de dispersão da quantidade de turnos por local

8.2.5 Gráfico da ordem de visita

Esse gráfico de barras exibe todas as ordens de visita aos locais durante as simulações. A Figura 53 apresenta um exemplo com cinco locais, em que a sequência de visita aos locais mais comum foi 1,3,4 e 5. Quantos mais locais e ligações entre eles houverem, maior será a possibilidade de caminhos.

A partir desse gráfico é possível visualizar a ordem de visitação dos locais que ocorreu com mais e menos frequência. A análise desse gráfico se mostra bem útil para avaliar as estratégias de movimentação ou como se comportam as simulações em que o *designer* faz uso de cadeados e chaves no mundo de jogo.

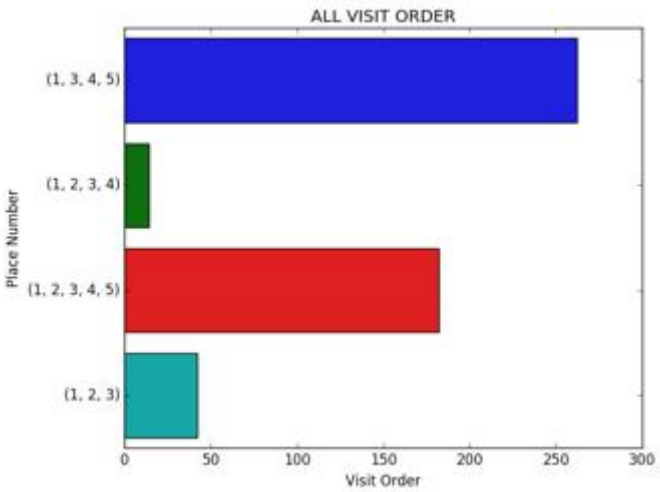

Figura 53: Gráfico com ordem de visita aos locais

8.2.6 Gráfico de sucesso e falha

O gráfico de sucesso e falha é o mais simples de todos, apenas consolida a porcentagem de sucesso e falha da equipe de heróis em um gráfico circular. É útil para medir o nível de dificuldade do jogo. A figura a seguir apresenta um exemplo de jogo com dificuldade elevada, em que a equipe de heróis obteve sucesso em somente 9,8% das simulações.

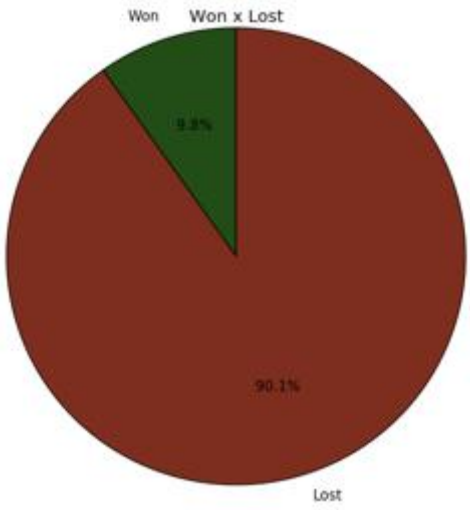

Figura 54: Gráfico de sucesso e falha

9. Avaliação

Esse capítulo inicia com a validação do simulador Pegasus, em seguida apresenta a experiência de uso com o Pegasus para apoiar o *design* de levels de alguns tipos de jogos de progressão. As próximas seções, além da validação do simulador, introduzem o tipo de jogo, construção dos artefatos de entrada, resultado de simulações e possibilidades de variações do jogo.

Os arquivos XML relacionados às simulações que representam os artefatos de entrada podem ser consultados no Anexo III. Alguns dos gráficos gerados e trechos de relatórios estão presentes nesse capítulo, todos os demais estão disponíveis no mesmo anexo.

9.1 Validação

Para validar o funcionamento do simulador Pegasus foram construídos e executados uma série de casos de teste, que visam alcançar as partes críticas de execução de código. Foram criados cenários a partir da disposição de locais e, sobre estes, uma série de fluxos com variações de heróis, monstros, itens, chaves e cadeados.

Os cenários criados estão listados na Tabela 9.

Tabela 9: Cenários de teste

Cenário	Qtd. de locais	Qtd. de locais finais	Possibilidade de caminhos
1	1	1	Único
2	2	1	Único
3	3	1	Único
4	3	1	2
5	3	2	2
6	4	1	Único
7	4	1	2
8	4	1	3
9	4	2	2
10	4	2	3
11	4	3	3

A Tabela 10 contém os fluxos de testes, para validar os resultados de variações sobre os cenários.

Tabela 10: Fluxos de teste

Fluxo	Heróis	Monstros	Chave	Itens
1	1	Não	Não	Não
2	1	Não	Sim	Não
3	2	Não	Não	Não
4	2	Não	Sim	Não
5	1	Fracos	Não	Não
6	1	Fortes	Não	Não
7	2	Fracos	Não	Não
8	2	Fortes	Não	Não
9	2	Fracos	Sim	Não
10	2	Fortes	Sim	Não
11	1	Fracos	Não	Ataque
12	1	Fortes	Não	Ataque
13	1	Fracos	Não	Defesa
14	1	Fortes	Não	Defesa
15	1	Fracos	Não	Agilidade
16	1	Fortes	Não	Agilidade
17	1	Fracos	Não	Vida
18	1	Fortes	Não	Vida
19	2	Fracos	Sim	Todos
20	2	Fortes	Sim	Todos

Os fluxos de teste são aplicados aos cenários de acordo com suas possibilidades. Por exemplo, não é viável a aplicação de um fluxo com sistema de chave e cadeado ao cenário que possui apenas um local.

A matriz a seguir (Tabela 11) demonstra a combinação de cenário com fluxo. As representações com o símbolo ✓ indicam que o resultado esperado é que o herói consiga chegar com sucesso ao local final superando os desafios. O símbolo X indica que o resultado esperado é que o herói não consiga alcançar seu objetivo.

Tabela 11: Matriz de resultados esperados dos casos de teste

Fluxo/Cenário	1	2	3	4	5	6	7	8	9	10	11
1	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
2		✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
3	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
4		✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
5	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
6	X	X	X	X	X	X	X	X	X	X	X
7	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
8		X	X	X	X	X	X	X	X	X	X
9		✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
10		X	X	X	X	X	X	X	X	X	X
11		✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
12		X	X	X	X	X	X	X	X	X	X
13		✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
14		X	X	X	X	X	X	X	X	X	X
15		✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
16		X	X	X	X	X	X	X	X	X	X
17		✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
18		X	X	X	X	X	X	X	X	X	X
19		✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
20		X	X	X	X	X	X	X	X	X	X

Nos fluxos que possuem monstros fortes, seus atributos possuem valores extremamente altos para alcançar o resultado esperado de heróis morrendo antes de chegar a um dos locais finais, mesmo nos fluxos em que existem itens que melhoram algum dos atributos. O intuito desses casos de testes é testar o insucesso dos heróis.

Os fluxos de 5 a 20 possuem batalhas. Para estes, os eventos de cada turno foram checados em tempo de execução de código e, posteriormente, confirmados no relatório

detalhado. O mesmo procedimento foi executado para checar a influência dos itens nos fluxos de 11 a 20.

A aferição dos resultados esperados de todos os casos de teste valida a ferramenta e passa a segurança de que sua execução segue de acordo com o imaginado durante sua concepção.

As seções a seguir apresentam a experiência de uso do simulador Pegasus no *design* de levels de jogos.

9.2 Jogo estilo aventura

O primeiro jogo apresentado é o mais abundante, contendo todos os elementos disponíveis para a construção do mundo de jogo: itens de *power-up* para ataque, defesa, vida, objetivo principal, mais de uma possibilidade de caminho, área inacessível opcional que pode ser desbloqueada com o uso de uma chave.

A ideia imaginada para esse exemplo seria um *level* de um jogo de aventura. Nesse *level*, uma equipe composta por três heróis (*Knight*, *Amazon* e *Wizard*) com habilidades distintas precisam chegar até o final do mapa de jogo e derrotar o chefão ali presente. Os heróis não possuem conhecimento prévio do mapa de jogo e uma vez que avançam, não é possível retornar ao local anterior. Mais de um caminho pode levar até o local final.

O mundo de jogo imaginado é composto de doze locais, a maioria com inimigos a serem enfrentados. Os locais números 2, 6 e 10 possuem itens para melhorar o ataque. O número 5 contém item para melhorar a defesa. Os locais 7 e 10 possuem itens que melhoram a vida. O local 10 possui um cadeado que só permite que este seja acessado se a chave, presente no local 5, for coletada. O local 12 é o último e contém um chefão que precisa ser derrotado. A Figura 55 apresenta o mapa do jogo descrito.

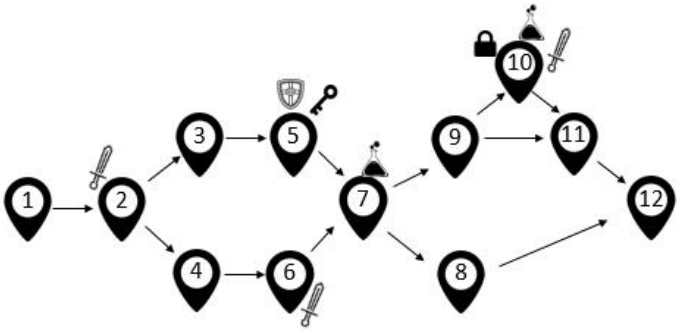

Figura 55: Mapa de jogo estilo aventura

A Tabela 12 mostra os inimigos e itens em cada local. O valor do item é apresentado entre parênteses. A Tabela 13 mostra os atributos dos heróis e inimigos. Finalmente, a Tabela 14 contém as configurações escolhidas para as simulações.

Tabela 12: Inimigos e itens por local

Local	Inimigo	Item
1	2 Goblins	-
2	1 Orc	Espada de ataque (5)
3	2 Goblins, 1 Orc	-
4	2 Goblins, 1 Troll	-
5	1 Goblin, 2 Trolls	Escudo de defesa (10), Chave de prata
6	1 Goblin, 2 Orcs	Espada de ataque (10)
7	1 Orc, 1 Troll	Poção de vida (80)
8	8 Goblins	-
9	2 Goblins, 1 Orc, 1 Troll	-
10	2 Trolls	Cadeado de prata, Poção de vida (80), Espada de ataque (8)
11	4 Goblins	-
12	Warlock (Chefão)	-

Tabela 13: Atributos dos personagens

Nome	Vida	Ataque	Defesa	Agilidade
Knight	100	10	8	10
Amazon	80	12	8	8
Wizard	120	14	5	5
Goblin	25	6	2	10
Orc	50	13	6	11
Troll	80	14	8	4
Warlock	200	18	9	5

Tabela 14: Configurações de simulação

Configuração	Opção escolhida
Estratégia de movimentação	Próximo aleatoriamente
Personagem para atacar	Aleatoriamente
Personagem para se beneficiar de item	Aleatoriamente
Mecânica de batalha	*Active Time Battle*
Gerador de aleatoriedade para ataque	Mínimo 0% e máximo 60%
Gerador de aleatoriedade para defesa	Mínimo 0% e máximo 30%

Todas as simulações desse trabalho foram executadas em um computador com processador Intel Core i5 da 5ª geração, 8GB RAM.

Foram executadas dez mil simulações em um tempo de treze minutos e seis segundos. A síntese do relatório indica que os heróis obtiveram sucesso 4.885 vezes (48,85%); o número médio de locais visitados por simulação foi 7,48; a sequência de visita mais ocorrida foi 1,2,3,4,5,7,8,12 por 2.451 vezes (24,51%). A seguir, a Figura 56 reúne alguns dados interessantes.

A parte da figura destacada com D mostra a porcentagem equilibrada de sucesso, o que indica que o *level* construído apresenta uma dificuldade média. Nesse momento o *designer* pode realizar alterações com base nos dados disponíveis para alterar o jogo de forma que fique de acordo com seu interesse. Por exemplo, se o *designer* quiser que o jogo se torne mais difícil as alterações deverão resultar numa porcentagem de sucesso menor.

A parte da figura destacada com A indica que na maioria das vezes a equipe de conseguiu chegar no último local da simulação. Uma maneira de aumentar a dificuldade é diminuindo esse número. A parte da figura destacada com C mostra que as batalhas nos locais de 1 a 7 possuíram poucos turnos, o que pode indicar poucos inimigos ou inimigos fáceis. Observando a parte da figura destacada com B é possível verificar que o locais de 3 a 7 fizeram a vida dos heróis diminuir consideravelmente, então estes já possuem um alto grau de dificuldade. No local 8 a vida dos heróis aumenta, pois no local anterior existe um item que concede vida. No entanto, o local 9 também é imediatamente após o 7 e não apresenta tanto aumento, indicando que o local 8 parece ter menos dificuldade que o local 9.

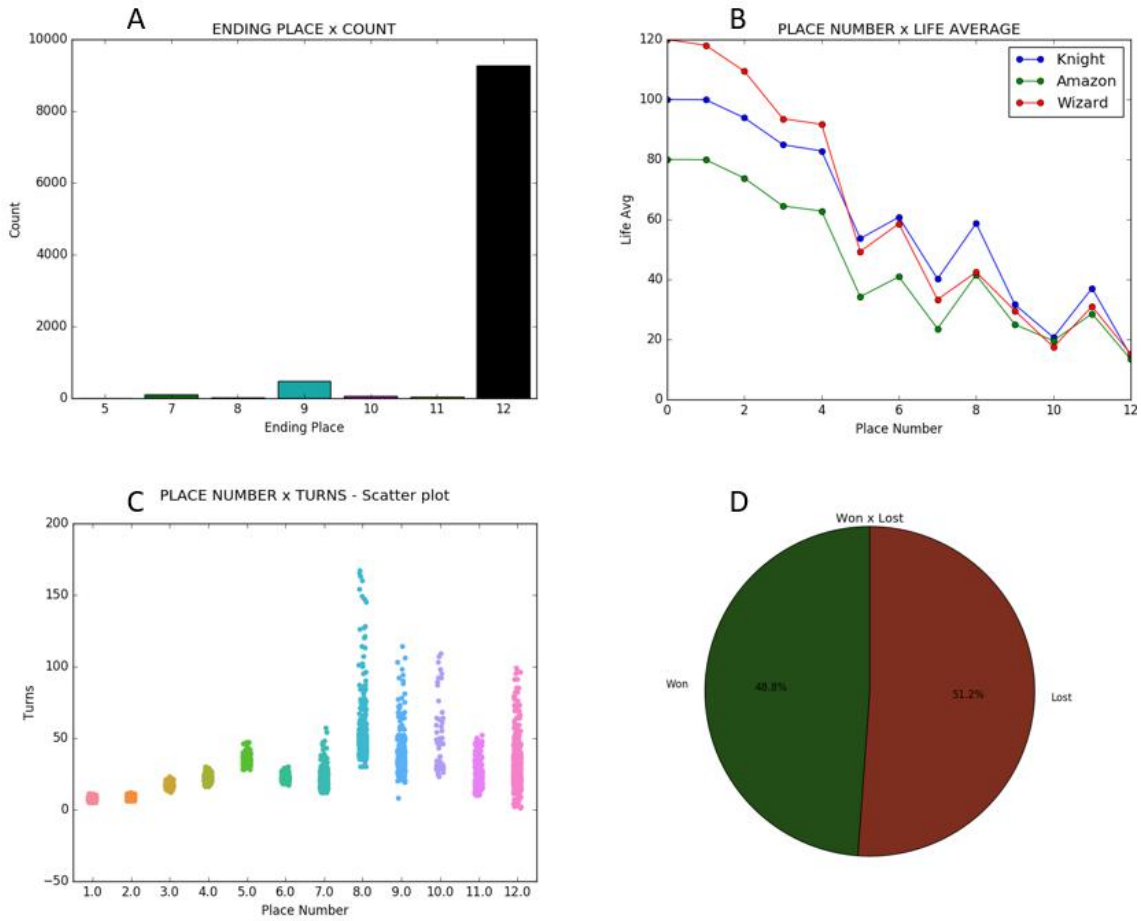

Figura 56: Alguns gráficos resultantes das simulações

Criar uma variação que resulte em aumento de dificuldade pode ser feito de diversas formas, como retirar alguns itens de *power-up*, aumentar algum atributo dos inimigos, aumentar quantidade de inimigos, etc. O *designer* deve executar sucessivamente o *Ciclo de testes* com apoio do simulador conforme descrito na proposta dessa tese até que o resultado seja o esperado.

A partir da análise descrita, a opção escolhida é aumentar a quantidade de inimigos nos locais 1 e 2, e incluir um inimigo de força mais elevada no local 8. A Tabela 15 mostra as modificações realizadas para a próxima bateria de simulações.

Tabela 15: Modificações para nova simulação

Local	**Inimigos antes**	**Inimigos depois**
1	2 Goblins	6 goblins
2	1 Orc	1 Goblin, 2 Orc
8	8 Goblins	8 Goblins, 1 Troll

Após uma sequência de outras dez mil simulações, em dez minutos e quatorze segundos, a síntese do relatório indica que os heróis obtiveram sucesso em 433 vezes (4,33%); houveram 5,78 locais visitados na média; a sequência de visita mais ocorrida foi 1, 2, 4, 6, 7 por 2.310 vezes (23,1%). A Figura 57 apresenta alguns dos novos dados.

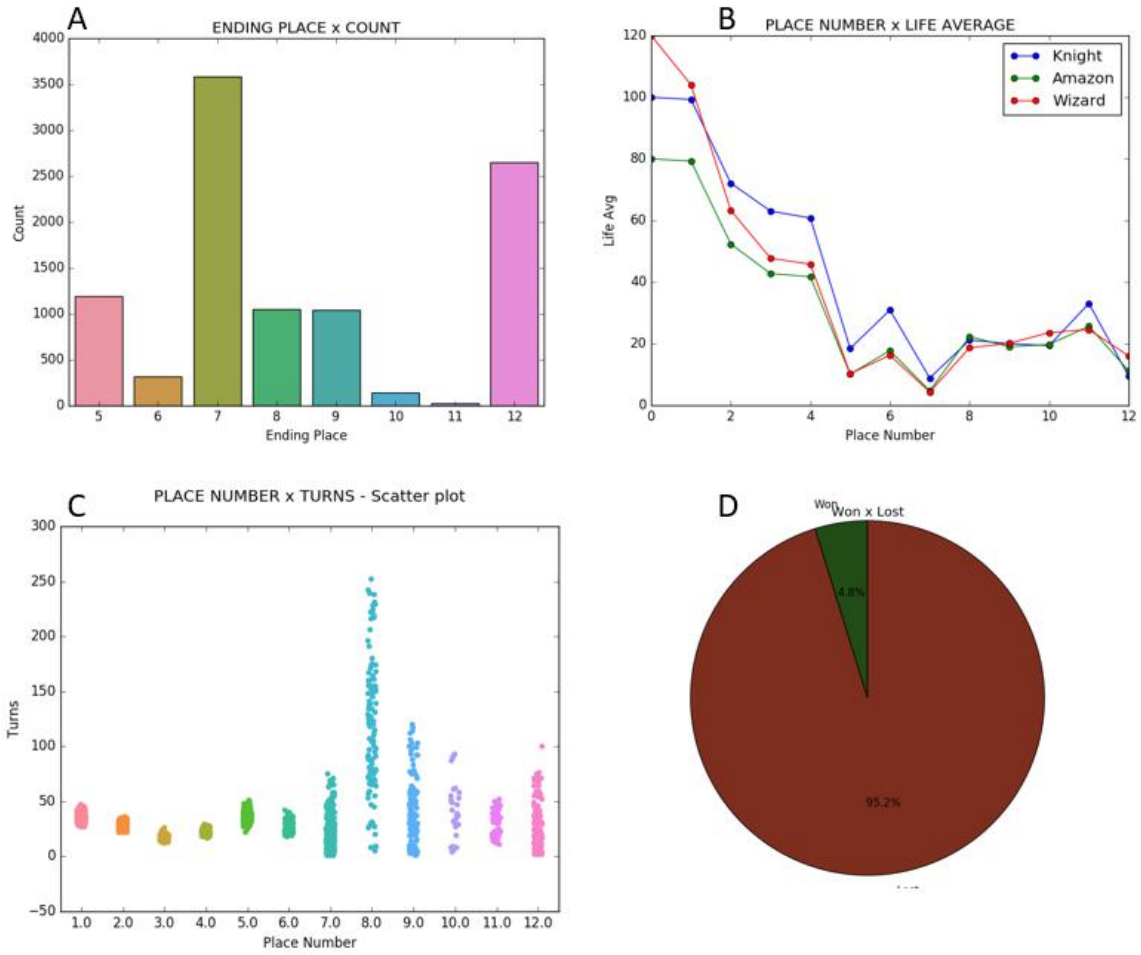

Figura 57: Gráficos resultantes das novas simulações

A parte da figura destacada com C mostra o aumento no número de turnos pretendido. Em A pode ser notado que o local 7 se tornou o local onde a simulação terminou mais vezes. Isso pode ter acontecido pelo fato de ser um local de passagem obrigatória e, após as modificações realizadas, os heróis chegam a esse local com menos vida. A parte D mostra que o *level* de jogo modelado passou a ter uma dificuldade bem mais elevada. A partir do local 5 a equipe de heróis tem uma média de vida reduzida em relação ao observado nas simulações anteriores, conforme pode ser visto em C.

Estes exemplos foram construídos para que o leitor possa avaliar o uso do simulador como ferramenta de apoio ao *designer* de jogos. Caso esse não seja o efeito desejado, cabe ao *designer* realizar novas alterações e continuar com o *Ciclo de testes* até que seja alcançado o resultado esperado.

Durante os testes do simulador, foi percebido que a variação da estratégia de escolha do personagem a atacar em batalhas pode produzir resultados distintos em relação a estratégia de escolha aleatória. Possivelmente a estratégia de escolha aleatória é ideal para avaliar o comportamento de um jogador comum. O ajuste de estratégias pode dar uma ideia de como jogadores mais habilidosos podem se comportar. Consequentemente, essa funcionalidade se mostra proveitosa para descobrir as melhores configurações para múltiplos níveis de dificuldade de um mesmo jogo.

9.3 Jogo com exploração

O segundo jogo apresentado procura demonstrar elementos de exploração, isto é, a possibilidade de navegar pelo mundo de jogo e revisitar locais. Essa característica é comum em jogos em que existem *puzzles* a serem resolvidos e itens que devem ser coletados a fim de desbloquear a passagem para o caminho principal.

Para avaliar esta característica de exploração, foi construído um jogo inspirado na no mapa da delegacia de polícia de *Resident Evil 2*. A Figura 58 presenta o mapa dos dois andares da delegacia de polícia.

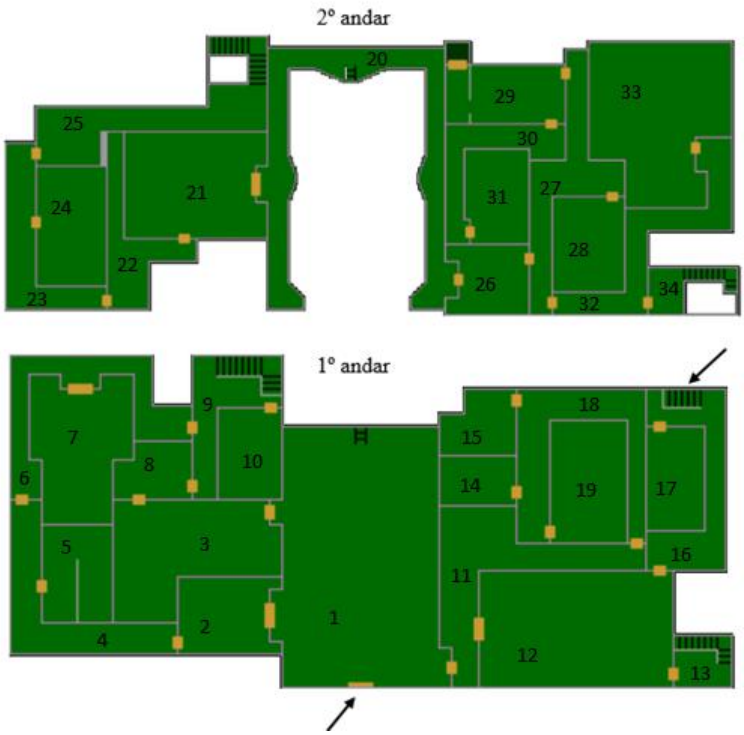

Figura 58: Mapa da delegacia de polícia de *Resident Evil 2*. Adaptado de (TAYLOR, 2018)

As marcações em amarelo representam as portas e as listras paralelas em preto são escadas. A seta mais ao centro aponta o local por onde o protagonista do jogo entra na delegacia. A seta mais à direita aponta as escadas que levam ao subterrâneo, que representa a saída da delegacia. E, por fim, os números não significam nenhuma ordem específica, eles foram acrescentados pelo autor desta tese para identificar cada um dos locais onde o jogo se passa.

Ao entrar na delegacia, muitas portas se encontram trancadas. É necessário enfrentar zumbis, monstros, encontrar uma série de chaves e itens até que o caminho considerado principal esteja desbloqueado para o herói poder avançar no jogo.

A Figura 59 mostra a construção de um mapa correspondente para o simulador. Nela, se encontram em destaque as áreas de cada andar, itens de chave e cadeados. A saída ao subterrâneo está representada como um local adicional, condicionando o local de término da simulação.

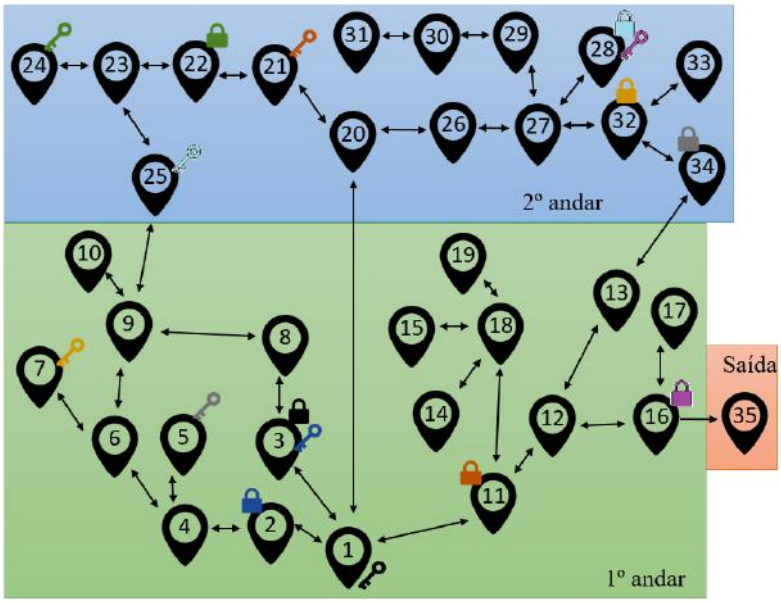

Figura 59: Mapa para delegacia de *Resident Evil 2*

Uma construção mais completa envolveria adicionar inimigos e itens de *power-ups* por alguns locais. No exemplo apresentado, a intenção é demonstrar a elaboração de um modelo de jogo com elementos de exploração e resolução de *puzzles*. Como nenhum inimigo foi incluído, o herói tem sucesso em 100% das simulações. Todas terminam no local número 35.

Foram realizadas mil simulações, com a estratégia de movimentação *novos locais*. Devido à ausência de batalhas, não cabe apresentar os gráficos gerados pela ferramenta. As informações completas dessas simulações se encontram no Anexo III. É interessante analisar algumas simulações do relatório resumido para verificar a ordem em que as chaves foram encontradas. A partir da síntese pode ser observado que a ordem de visita mais comum passou por 338 locais, acontecendo por apenas 2 vezes (0,2%).

A estratégia de movimentação foi alterada para *próximo aleatoriamente* e um novo grupo de simulações não produziram resultados com diferenças significativas.

A imensa quantidade de locais e conexões possibilitam inúmeras combinações de caminhos possíveis. Esse cenário, aliado a oito sistemas de chave e cadeado, e mais uma estratégia de movimentação fraca, foi determinante para que a sequência de visita mais comum obtivesse um número tão baixo. Esse número pode ser melhorado se forem criadas estratégias mais adequadas a esse cenário.

O ambiente de simulação, ainda assim, consegue apoiar o *designer* na representação visual e a emular o comportamento de jogadores. Em jogos desse estilo é

comum o jogador falar que “ficou empacado” em certa parte, o que significa que não encontrou alguma chave para liberar o caminho principal. Além disso, o fato de todas as simulações terem conseguido atingir o local final indica que o mapa elaborado possui uma solução viável, o que é o mais importante.

9.4 Jogos divididos em mundos ou etapas

Um estilo de jogo de progressão bastante encontrado é a divisão em grandes levels, cada um constituído por pequenos *levels*. A seguir são mostrados alguns jogos conhecidos com essa característica de *design* e exemplos de construção de mundo de jogo para simular tais ambientes.

9.4.1 Super Mario Bros 3

O jogo *Super Mario Bros 3* foi lançado originalmente no Japão em 1988 para o *Nintendo Entertainment System*, chegando nos Estados Unidos somente dois anos mais tarde (NINTENDO, 2018). O console ficaria conhecido no Brasil como *Nintendinho* após o lançamento do *Super Nintendo*. Não há dúvidas de que o personagem Mario é um dos mais populares do mundo do videogame.

O *Super Mario Bros 3* é composto de 8 mundos. Em cada mundo existem várias fases e no final um castelo com chefão. A Figura 60 contém o mapa e legenda do primeiro mundo.

Figura 60: Primeiro mundo de *Super Mario Bros 3*. Adaptado de (VGMAPS, 2013)

A fortaleza e o portão são claramente elementos de chave e cadeado. No mapa do primeiro mundo o caminho desbloqueado não concede nenhuma vantagem, mas serve de

ensinamento para o jogador saber como funciona esse sistema e utilizar nos outros mundos. As fases 3 e 4 são opcionais. Passar por uma delas concede acesso à casa de cogumelo, que recompensa o jogador com item.

A figura a seguir mostra a construção do mundo de jogo correspondente para o simulador. O local 1 corresponde ao *start*; os locais 2, 3, 4, 5, 10 e 11 correspondem às fases de 1 a 6 no jogo original; os locais 6, 7 e 12 fornecem itens; a fortaleza e o portão estão representados nos locais 8 e 9, respectivamente; o local 13 é o local final, com o castelo onde reside o chefão do mundo.

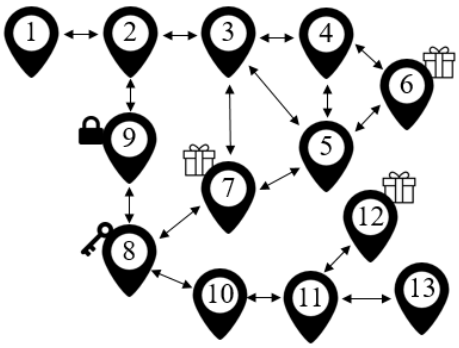

Figura 61: Mapa para o primeiro mundo de *Super Mario Bros 3*

É possível também construir um modelo com todos os mundos de *Super Mario Bros 3*, incluindo conexões entre o local final de um mundo e o local inicial do mundo seguinte. A Figura 62 ilustra esse cenário.

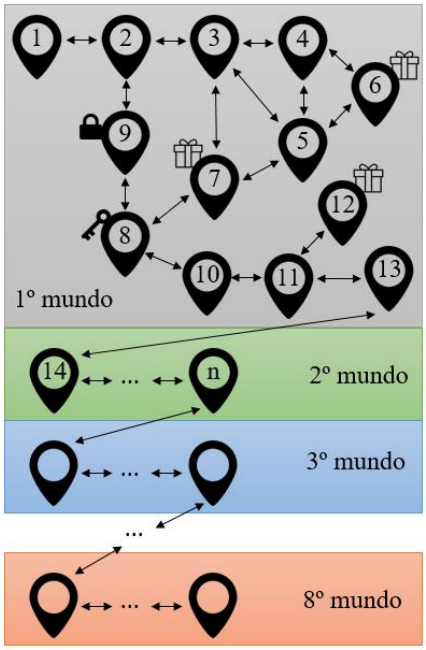

Figura 62: Representação dos 8 mundos de *Super Mario Bros 3*

Um item bastante popular deste jogo são as flautas secretas que permitem teletransportar o personagem principal para outros mundos, permitindo avançar mais rápido no jogo. A funcionalidade das flautas pode ser representada com a inclusão de locais, inicialmente trancados, entre os mundos. A flauta seria a chave para desbloquear esses locais de passagem que conectam os mundos.

9.4.2 Diablo

Diablo é um jogo de RPG com elementos de fantasia e batalhas lançado pela *Bizzard* em 1996 (BLIZZARD, 2018). O jogador controla um herói e vai literalmente descendo andares de uma espécie de masmorra ou labirinto, enfrentando vários inimigos e coletando itens de *power-ups*, peças de ouro que servem para comprar outros itens, chaves etc. Cada andar corresponde a um *level*, seus elementos são gerados aleatoriamente, porém conforme vai descendo a dificuldade vai aumentando. Ao final de 16 *levels* o jogador enfrenta o próprio *Diablo*.

O mundo de jogo de *Diablo* pode ser representado com conexões entre os *levels*, nos mesmos moldes do apresentado para os mundos de *Super Mario Bros 3*. Apesar da geração automática de inimigos e itens em cada *level*, é possível incluir inimigos em cada local para avaliar o crescimento de dificuldade no simulador. A Figura 63 contém o mapa construído.

Figura 63: Representação dos 16 *levels* de *Diablo*

Para simular *Diablo* foram construídos personagens de herói, monstro e o Diablo. O primeiro *level* inicia com dois monstros e a cada *level* essa quantidade aumenta em dois. Sempre que um *level* é par existe um item de *power-up* como recompensa. A Tabela 16 mostra os atributos dos personagens.

Tabela 16: Atributos dos personagens para *Diablo*

Nome	Vida	Ataque	Defesa	Agilidade
Hero	100	10	8	10
Monster	20	6	2	6
Diablo	300	16	8	4

As configurações de simulação utilizadas são as mesmas da Tabela 14. Após mil simulações em um tempo de três minutos e nove segundos a síntese do relatório aponta que o mais longe que o herói conseguiu atingir foi o *level* 12, indicando que o *design* inicial deixou o jogo muito difícil. A Figura 64 contém alguns dos gráficos gerados.

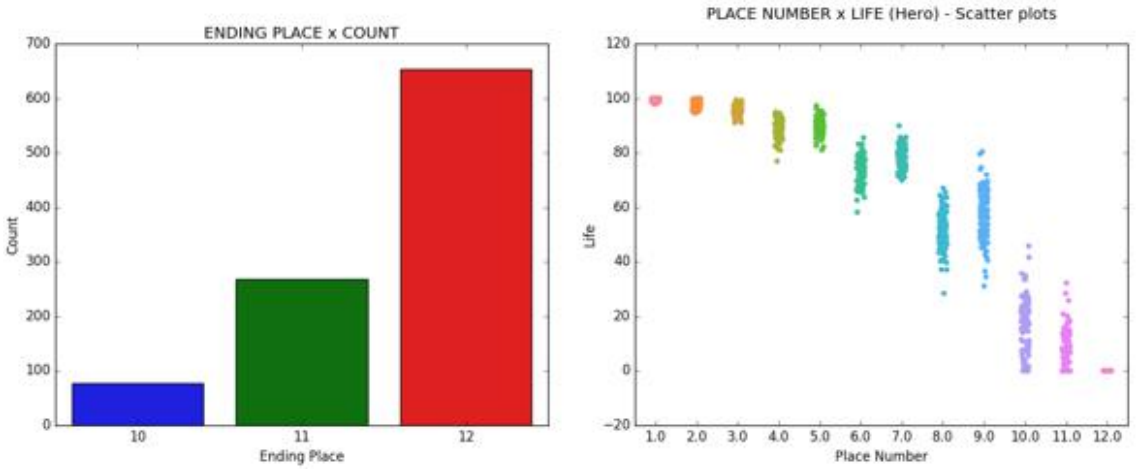

Figura 64: Gráficos de local final e vida do personagem por local

Para alterar esse cenário, foram adicionados *power-ups* nos locais que não possuíam e foram removidos um monstro de cada local.

Após uma nova sequência de mil simulações, em quatro minutos e vinte e um segundos, a síntese do relatório mostra uma taxa de 0,7% de sucesso. A figura a seguir apresenta a variação da vida do herói por local ao longo das simulações.

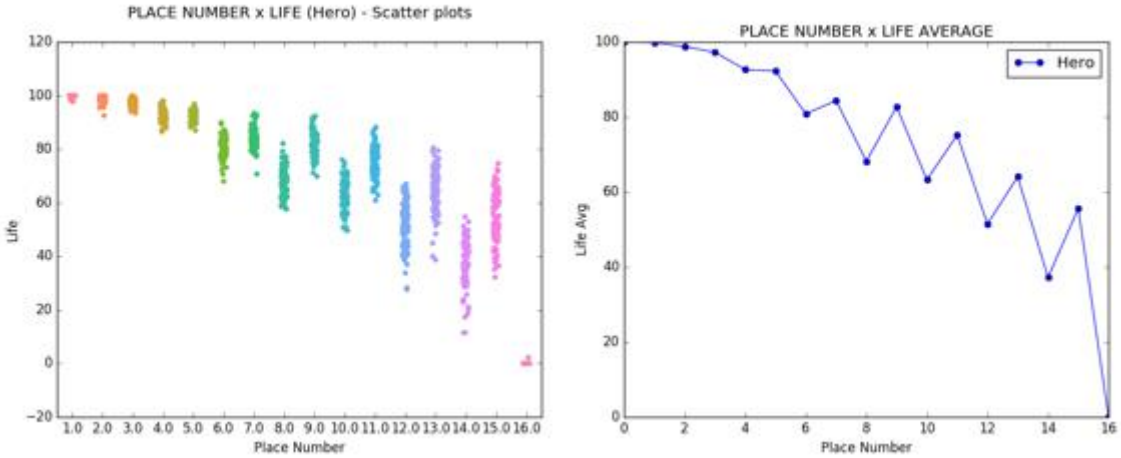

Figura 65: Variação da vida do herói e valor médio por local

Em todas as simulações foi atingido o último local, e mesmo assim somente por sete vezes o chefão final foi derrotado. A conclusão é que o chefão está com dificuldade elevada. O *designer* pode realizar modificações e continuar com o *ciclo de testes* ou pode dar por encerrada esta fase.

9.4.3 Outros jogos

Os jogos a seguir são bem conhecidos e também podem ser modelados da mesma forma que os apresentados nessa seção.

- *Sonic The Hedgehog (1991)*: 7 zonas, cada uma com 3 atos;
- *Donkey Kong Country (1994)*: 6 mundos, cada um com 5 ou 6 fases;
- *Crash Bandicoot 3 (1998)*: 6 zonas, cada uma com 5 fases;
- *Gears of War (2006)*: 5 atos, cada um com 6 ou 8 capítulos;
- *Alan Wake (2010)*: 6 episódios, cada um com 3 partes.

Uma característica marcante dos jogos de progressão é contar uma história. Quando o jogo causa empatia com o público e é bem desenvolvido, pode resultar em continuações, como ocorre com todos os citados acima.

10. Conclusões

Este capítulo final apresenta as conclusões, as principais contribuições da pesquisa, as limitações enfrentadas neste trabalho, bem como sugestões de trabalhos futuros.

10.1 Epílogo

Esta tese realiza um estudo na área de jogos, aborda o papel do game *designer* e discute a representação de jogos em modelos. O balanceamento do jogo é destacado como uma maneira de obter harmonia entre os elementos que o compõe, sendo a simulação uma das técnicas para realizar tal feito.

São conceituados jogos com estrutura de emergência e progressão e exposta a carência de ferramentas de apoio ao *designer* de jogo, sobretudo aos que fazem parte desse último estilo. São apresentados padrões para *design* de elementos de jogos comumente encontrados em jogos de progressão.

São relatadas experiências de representações de jogos com Redes de Petri de Alto-Nível e com Máquina de Estado. É proposto um ambiente para apoiar o *design* de jogos, são expostos os tipos de jogos abordados e as possibilidades de ganhos em *playability*. É apresentada a ferramenta de simulação Pegasus, sua arquitetura em camadas e componentes auxiliares, construídos para avaliar a proposta.

É relatada a experiência de uso do simulador com exemplos de jogos modelados, simulados, atuação do *designer* de jogo e, por fim, modelagem de alguns jogos reais.

10.2 Contribuições

O *design* de jogos constitui o principal foco de estudo dessa tese. Ela propõe um ambiente de apoio ao *design* de jogos de progressão, para solucionar o problema da falta ferramentas de auxílio ao *designer*. Nesse sentido, a expectativa é de contribuir com uma proposta inédita que traga resultados inovadores e signifique, portanto, um avanço no estado da arte dessa área de aplicação.

O Pegasus demonstrou potencial em se tornar uma valiosa ferramenta de apoio o *designer* de jogo em suas tarefas. A habilidade em testar variações de cenários, ainda

durante a fase de elaboração de um jogo, é um grande benefício que pode resultar em economia de tempo e custo.

Ao final deste trabalho, as principais contribuições podem ser relacionadas, estando em conformidade com os objetivos propostos:

- Estudo sobre jogos, representação de jogos, *design* e padrões para *design* de jogos com estrutura de progressão;
- Elaboração detalhada de uma proposta de ambiente de simulação para jogos com estrutura de progressão;
- Implementação do Pegasus, um simulador desenvolvido de acordo com padrões de projeto de *software* e arquitetura com facilidade de expansão;
- Implementação de componentes auxiliares para facilitar a utilização do simulador;
- Avaliação da proposta mediante construção e simulação de modelos de jogos no Pegasus.

10.3 Limitações e trabalhos futuros

A proposta elaborada não cobre todos os aspectos dos jogos. Arte, inteligência artificial e mecânica são algumas das outras áreas relevantes em *design* de jogos que não são discutidas nessa tese. O foco dessa tese são os jogos de progressão, mas mesmo estes têm a fidelidade de sua representação em modelo condicionada a construção com os elementos de jogos previstos.

O sistema de batalhas é baseado em ataques corpo-a-corpo, o que faz com que jogos que possuem ataques a distância não sejam fielmente simulados. No entanto, essa seria uma evolução do sistema como trabalho futuro, uma vez que o arcabouço que circunda os eventos de batalha se encontra construído.

Durante os testes com o Pegasus foi observado que as estratégias de movimentação funcionam bem para mapas de mundo de jogo com características lineares. Conforme aumenta a quantidade de caminhos, maior a chance de os heróis ficarem revisitando locais passados, em círculos, e demorarem para atingir o local final. Um dos trabalhos futuros é o desenvolvimento de novas estratégias de movimentação, aliadas a objetivos e o sistema de chave e cadeado.

Como o simulador possui uma arquitetura de fácil expansão, os trabalhos futuros sugeridos são baseados no aprimoramento da experiência de uso do simulador.

Um possível trabalho futuro é a expansão da Camada de Processamento de Resultado, para gerar novos gráficos a partir das informações armazenadas na Camada de Armazenamento de Resultado. Outro possível trabalho consiste em desenvolver novas opções para os módulos da Camada de Configuração e Integração, e desenvolver novos componentes auxiliares.

Por fim, um outro trabalho futuro seria o aprimoramento do ambiente de simulação para que o *designer* possa, de uma só vez, preparar mais de um cenário e as simulações gerariam coleções de resultados. Essa melhoria traria como benefício a diminuição da quantidade de iterações do *designer* durante o *Ciclo de testes*, porém aumentando possivelmente o tempo de simulação.

Referências Bibliográficas

ADAMS, E., 2009, *Fundamentals of Game Design*. . 2nd. Thousand Oaks, CA, USA, New Riders Publishing.

ADAMS, E., DORMANS, J., 2012, *Game Mechanics: Advanced Game Design*. . S.l., New Riders.

ADOBE, 1996. Disponível em: <https://get.adobe.com/br/flashplayer/>. Acessado em: 9 Janeiro 2018.

ALMEIDA, F. DE Q.B., 2015a. *RACHINATIONS: Modelando a Economia Interna de Jogos*. . S.l.: Universidade Federal do Rio de Janeiro.

ALMEIDA, F. DE Q.B., 2015b. Disponível em: <https://github.com/queirozfcom/rachinations>. Acessado em: 10 Maio 2015.

ANDITYA, A., SIHABUDDIN, A., 2016. "An optimal input for Role-Playing Game's combat pace using an Active Time Battle system algorithm". In: *2016 2nd International Conference on Science and Technology-Computer (ICST)*. S.l.: s.n. 2016. pp. 61–65.

ANDRADE, G., RAMALHO, G., SANTANA, H., et al., 2005, "Automatic computer game balancing". In: *Proceedings of the fourth international joint conference on Autonomous agents and multiagent systems - AAMAS '05*. pp. 1111.

ARAÚJO, M., ROQUE, L., 2009, "Modeling Games with Petri Nets". In: *Digital Games Research Association (DiGRA)*.

ASHLOCK, D., 2010. "Automatic generation of game elements via evolution". In: *Computational Intelligence and Games CIG 2010 IEEE Symposium on*. S.l.: Ieee. 2010. pp. 289–296.

BARBOSA, A.F.S., PEREIRA, P.N.M., DIAS, J.A.F.F., et al., 2014, "A New Methodology of Design and Development of Serious Games". In: *International Journal of Computer Games Technology*. v. 2014, pp. 1–8.

BARRETO, N., 2013. *A Language for Game Design and Coreography*. . S.l.: Universidade de Coimbra.

BELLOTTI, F., BERTA, R., DE GLORIA, A., et al., 2012, "A Serious Game Model for Cultural Heritage". In: *Journal on Computing and Cultural Heritage*. v. 5, pp. 1–27.

BETHKE, E., 2003, *Game development and production*. . Plano, Texas, Wordware Publishing, Inc.

BJÖRK, S., HOLOPAINEN, J., 2005, *Patterns in Game Design*. . S.l., Charles River

Media.

BLIZZARD, 2018. Disponível em: <http://us.blizzard.com/en-us/games/legacy/>. Acessado em: 7 Fevereiro 2018.

BNDES, 2018. Disponível em: <https://www.bndes.gov.br/wps/portal/site/home/conhecimento/noticias/noticia/jogos-digitais-brasil-infografico>. Acessado em: 29 Janeiro 2018.

BOARDGAMEGEEK, 2018. Disponível em: <https://boardgamegeek.com/browse/boardgamecategory>.

BROM, C., ABONYI, A., 2006. "Petri-Nets for Game Plot". In: *Proceedings of Artificial Intelligence and Simulation Behaviour (AISB)*. S.l.: s.n. 2006.

BROWNE, C., 2008. *Automatic Generation and Evaluation of Recombination Games*. . S.l.: Queensland University of Technology.

BROWNE, C., 2011, *Evolutionary Game Design*. . London, Springer London. Acessado em: 2 Setembro 2015. SpringerBriefs in Computer Science.

BURA, S., 2006. *A Grammar Game*. S.l. Disponível em: <http://www.stephanebura.com/diagrams/>.

BURGUN, K., 2011. GAMASUTRA - UNDERSTANDING BALANCE IN VIDEO GAMES. Disponível em: <https://www.gamasutra.com/view/feature/134768/understanding_balance_in_video_.php>. Acessado em: 1 Abril 2018.

CBS, 2015. Disponível em: <http://www.cbs.com/shows/big_bang_theory/>. Acessado em: 15 Julho 2015.

CLEARWATER, D., 2011, "What defines video game genre? Thinking about genre study after the great divide". In: *Loading...* v. 5, pp. 29–49.

COHEN, S.J., HUTCHINSON, A., 1954, "Labanotation". In: *The Journal of Aesthetics and Art Criticism*. v. 13, pp. 276.

COSTIKYAN, G., 1994. Disponível em: <http://www.costik.com/nowords.html>. Acessado em: 8 Abril 2015.

COSTIKYAN, G., 2002, "I Have No Words but I Must Design: Toward a Critical Vocabulary for Games". In: *Computer Games and Digital Cultures Conference*. pp. 9–33.

CPN 2000 PROJECT, 2003. Disponível em: <http://www.daimi.au.dk/CPnets/CPN2000/>. Acessado em: 20 Maio 2015.

CRAWFORD, C., 1984, *The art of computer game design*. . S.l., Osborne/McGraw-Hill. Acessado em: 3 Janeiro 2018.

CRAWFORD, C., 2003, *Chris Crawford on game design*. . S.l., New Riders. Acessado em: 29 Dezembro 2017.

VAN DAALEN, C.E., SCHAFFERNICHT, M., MAYER, I., 2014, "System Dynamics and Serious Games". In: *32nd International Conference of the System Dynamics Society*. pp. 1–26.

DANIEL BALAS, CYRIL BROM, ADAM ABONYI, J.G., 2008. "Hierarchical petri nets for story plots featuring virtual humans". In: *Proceedings of the Fourth Artificial Intelligence and Interactive Digital Entertainment Conference (AIIDE)*. S.l.: The AAAI Press. 2008. pp. 2–9.

DESIGN/CPN, 2004. Disponível em: <http://www.daimi.au.dk/designCPN/>. Acessado em: 20 Maio 2015.

DIZIKES, P., 2013. MIT NEWS MAGAZINE. Disponível em: <http://www.technologyreview.com/article/520181/the-beer-game/>. Acessado em: 27 Abril 2015.

DORMANS, J., 2012a. "The Effectiveness and Efficiency of Model Driven Game Design". In: *11th International Conference, ICEC 2012, Bremen, Germany, September 26-29, 2012. Proceedings*. S.l.: s.n. 2012. pp. 542–548.

DORMANS, J., 2012b, *Engineering Emergence: Applied Theory for Game Design*. . S.l., Universiteit van Amsterdam.

DZGOEVA, S., 2016. *Development of a Gamification Design Method for a Business Process Modeling Tool*. . S.l.: Westfälische Wilhelms-Universität Münster.

EINDHOVEN UNIVERSITY OF TECHNOLOGY, 2015. Disponível em: <http://cpntools.org/>. Acessado em: 25 Maio 2014.

ELETRONIC ARTS, 2014. Disponível em: <www.simcity.com>. Acessado em: 9 Abril 2015.

ENTERTAINMENT SOFTWARE ASSOCIATON, 2017. *Essential Facts About the Computer and Videogame Industry*. Washington, DC. Disponível em: <http://www.theesa.com/wp-content/uploads/2017/06/!EF2017_Design_FinalDigital.pdf>.

FAIRCLOUGH, C., FAGAN, M., MAC NAMEE, B., et al., 2001, "Research Directions for AI in Computer Games". In: *Irish Conference on Artificial Intelligence and Cognitive Science*. pp. 333–344.

FELDER, D., 2015. GAMASUTRA - DESIGN 101: BALANCING GAMES. Disponível em: <https://www.gamasutra.com/blogs/DanFelder/20151012/251443/Design_101_Balancing_Games.php>. Acessado em: 1 Abril 2018.

FORD, D.N., 1998. "System Dynamics as a Strategy for Learning to Learn". . 1998.

S.l.: System Dynamics Society.

FRASCA, G., 2003, "Simulation versus Narrative – Introduction to Ludology". In: *The Video Game Theory Reader*. v. 2003, pp. 221–235.

GAMMA, E., HELM, R., JOHNSON, R., et al., 1995, *Design Patterns: Elements of Reusable Object-Oriented Software*. . Boston, MA, USA, Addison-Wesley Longman Publishing Co., Inc.

GENESERETH, M., LOVE, N., PELL, B., 2005, "General game playing: Overview of the AAAI competition". In: *AI magazine*. v. 26, pp. 1–16.

GENESERETH, M., THIELSCHER, M., 2014, *General Game Playing*. . S.l., s.n.

GRÜNVOGEL, S.M., 2005, "Formal models and game design". In: *Game Studies: The International Journal of Computer Game Research*. v. 5.

GRÜNVOGEL, S.M., NATKIN, S., VEGA, L., 2004. "A new methodology for spatiotemporal Game Design". In: *Proceedings of CGAIDE*. S.l.: s.n. 2004. pp. 109–113.

HASBRO, 1935. Disponível em: <http://www.hasbro.com/en-us/product/monopoly-electronic-banking-game:EB2C42C9-5056-9047-F52D-5E3CC0532D6B>.

HIRSCH, G.B., HOMER, J.B., MCDONNELL, G., et al., 2005. "Achieving Health Care Reform in the United States: Toward a Whole-System Understanding". . 2005. S.l.: The System Dynamics Society.

HUNICKE, R., LEBLANC, M., ZUBEK, R., 2004, "MDA: A Formal Approach to Game Design and Game Research". In: *Workshop on Challenges in Game AI*. pp. 1–4.

IRISH, D., 2005, *The Game Producer's Handbook*. . Boston, MA, United States, Course Thomson Technology.

JAFFE, A., MILLER, A., ANDERSEN, E., et al., 2012, "Evaluating Competitive Game Balance with Restricted Play". In: *Proceedings 8th Artificial …*. pp. 26–31.

JÄRVINEN, A., 2009. *Games without Frontiers: Theories and Methods for Game Studies and Design*. . S.l.: University of Tampere, Finland.

JENSEN, K., 1996, *Coloured Petri Nets: Basic Concepts, Analysis Methods and Practical Use*. . S.l., Springer. EATCS monographs on Theoretical Computer Science.

JENSEN, K., 2015. Disponível em: <http://cs.au.dk/CPnets/>. Acessado em: 20 Maio 2015.

JENSEN, K., KRISTENSEN, L.M., 2009, "Coloured Petri Nets: Modelling and Validation of Concurrent Systems". In: *Springer*. v. 9, pp. 384.

JUUL, J., 2002. "The Open and the Closed: Games of Emergence and Games of Progression". In: *Computer Games and Digital Cultures Conference Proceedings*. S.l.: Tampere University Press. 2002.

JUUL, J., 2005, *Half-real : video games between real rules and fictional worlds*. . S.l., MIT Press. Acessado em: 7 Março 2018.

KAAS, S.J., 2012. Disponível em: <http://www.samkass.com/theories/RPSSL.html>. Acessado em: 15 Julho 2015.

KAISER, D.M., 2005. "The structure of games". In: *Proceedings of the 43rd annual southeast regional conference on - ACM-SE 43*. S.l.: ACM Press. 2005. pp. 61.

KLINT, P., ROZEN, R. VAN, 2013. "Micro-Machinations". In: ERWIG, Martin, PAIGE, Richard F. & VAN WYK, Eric (eds.), *Software Language Engineering*. Cham: Springer International Publishing. Lecture Notes in Computer Science. pp. 36–55.

KOSTER, R., 2005a. "A Grammar of Gameplay: Game Design Atoms, Can Games Be Diagrammed?". In: *Game Developers Conference*. S.l.: s.n. 2005.

KOSTER, R., 2005b, *Theory of Fun for Game Design*. . S.l., s.n.

KOSTER, R., 2012. "A Theory of Fun 10 years later". In: *Game Developers Conference*. S.l.: s.n. 2012.

KRIGLSTEIN, S., BROWN, R.A., WALLNER, G., 2014. "Workflow Patterns as a Means to Model Task Succession in Games: A Preliminary Case Study". In: PISAN, Yusuf, MARSCH, Tim & SGOUROS, Nikitas M. (eds.), *Entertainment Computing - ICEC 2014*. Sydney: Springer Berlin Heidelberg. pp. 36–41.

LIU, Y., WU, W., 2013. "Petri net modeling analysis of processes at clutch time in NBA games". In: *2013 IEEE International Conference of IEEE Region 10 (TENCON 2013)*. S.l.: IEEE. 2013. pp. 1–4.

LIVESCIENCE, 2011. Disponível em: <http://www.livescience.com/15574-win-rock-paper-scissors.html>. Acessado em: 15 Julho 2015.

MALLETT, J., LEFLER, M., 1998. Disponível em: <http://www.zillions-of-games.com>. Acessado em: 20 Maio 2015.

MCGONIGAL, J., 2010. Disponível em: <www.youtube.com/watch?v=dE1DuBesGYM>. Acessado em: 9 Abril 2015.

MCGONIGAL, J., 2011, *Reality Is Broken: Why Games Make Us Better and How They Can Change the World*. . S.l., s.n.

MCGONIGAL, J., 2012a. Disponível em: <www.youtube.com/watch?v=lfBpsV1Hwqs>. Acessado em: 7 Abril 2015.

MCGONIGAL, J., 2012b. Disponível em: <www.superbetter.com>. Acessado em: 9 Abril 2015.

MEGAJOGOS, 2015. Disponível em: <http://www.megajogos.com.br/jogosonline/rouba-monte/regras>.

MICROSOFT STUDIOS, 2011. Disponível em: <http://www.microsoft.com/games/flight/>. Acessado em: 7 Maio 2015.

MILNER, R., TOFTE, M., HARPER, R., et al., 1997, *The Definition of Standard ML - Revised*. . S.l., MIT Press.

NAMCO, 1983. Disponível em: <https://archive.org/details/msdos_Pac-Man_1983>. Acessado em: 8 Maio 2015.

NEIL, K., 2012. "Game design tools: Time to evaluate". In: *Proceedings of 2012 International DiGRA Nordic Conference*. S.l.: s.n. 2012.

NELSON, M., MATEAS, M., 2007. "Towards automated game design". In: *Proceedings of 10th Artificial Intelligence and Human-Oriented Computing (AI*IA 2007)*. S.l.: s.n. 2007. pp. 626–637.

NINTENDO, 2018. Disponível em: <https://www.nintendo.com/nes-classic/super-mario-bros-and-super-mario-bros-3-developer-interview>. Acessado em: 5 Fevereiro 2018.

NUMMENMAA, T., KUITTINEN, J., HOLOPAINEN, J., 2009. "Simulation as a game design tool". In: *Proceedings of the International Conference on Advances in Computer Enterntainment Technology - ACE '09*. Athens, Greece: ACM Press. 2009. pp. 232–239.

DE OLIVEIRA, G.W., JULIA, S., PASSOS, L.M.S., 2011. "Game modeling using WorkFlow nets". In: *Conference Proceedings - IEEE International Conference on Systems, Man and Cybernetics*. S.l.: s.n. 2011. pp. 838–843.

PETERSON, J.L., 1977, "Petri Nets". In: *ACM Computing Surveys*. v. 9, pp. 223–252.

PETERSON, J.L., 1978. "An Introduction to Petri Nets". In: *Proceedings of the National Electronics Conference, Volume 32*. S.l.: s.n. 1978. pp. 144–148.

PETERSON, J.L., 1981, *Petri Net Theory and the Modeling of Systems*. . S.l., Prentice-Hall.

PETRI, C.A., 1962. *Kommunikation mit Automaten*. . S.l.: Universität Hamburg.

PYGAME, 2014. Disponível em: <http://www.pygame.org>. Acessado em: 10 Abril 2014.

RARE, 1991. Disponível em: <https://archive.org/details/gg_Battletoads_1993Rare_Tradewest_SegaUS>.

Acessado em: 13 Janeiro 2018.

RATZER, A.V., WELLS, L., LASSEN, H.M., et al., 2003. "CPN Tools for Editing, Simulating, and Analysing Coloured Petri Nets". In: *Proceedings of the 24th international conference on Applications and theory of Petri nets (ICATPN'03)*. Eindhoven, The Netherlands: Springer-Verlag. 2003. pp. 450–462.

RIEMER, K., 2008. "The Beergame in business-to-business eCommerce courses – a teaching report". In: *21th Bled eConference eCollaboration: Overcoming Boundaries Through Multi-Channel Interaction*. S.l.: s.n. 2008. pp. 588–606.

RIEMER, K., 2012. Disponível em: <http://www.beergame.org/the-game>. Acessado em: 17 Abril 2015.

ROLLINGS, A., MORRIS, D., 2004, *Game Architecture and Design: A New Edition*. . Indianapolis, Indiana, New Riders Publishing.

ROZEN, R. VAN, DORMANS, J., 2014. "Adapting Game Mechanics with Micro-Machinations". In: *Foundations of Digital Games*. S.l.: Society for the Advancement of the Science of Digital Games. 2014.

RUBEN, B.D., 1999, "Simulations, Games, and Experience-Based Learning: The Quest for a New Paradigm for Teaching and Learning". In: *Simulation & Gaming*. v. 30, pp. 498–505.

SALEN, K., ZIMMERMAN, E., 2004, *Rules of Play: Game Design Fundamentals*. . S.l., The MIT Press Cambridge.

SALGE, C., MAHLMANN, T., 2010. "Relevant Information as a formalised approach to evaluate game mechanics". In: *Proceedings of the 2010 IEEE Conference on Computational Intelligence and Games (CIG)*. S.l.: IEEE. 2010. pp. 281–288.

SÁNCHEZ, J.L.G., VELA, F.L.G., SIMARRO, F.M., et al., 2012, "Playability: analysing user experience in video games". In: *Behaviour & Information Technology*. v. 31, pp. 1033–1054.

SÁNCHEZ, J.L.G., ZEA, N.P., GUTIÉRREZ, F.L., 2009. "Playability: How to Identify the Player Experience in a Video Game". In: GROSS, Tom, GULLIKSEN, Jan, KOTZ, Paula, OESTREICHER, Lars, PALANQUE, Philippe, PRATES, Raquel Oliveira & WINCKLER, Marco (eds.), *Human-Computer Interaction -- INTERACT 2009: 12th IFIP TC 13 International Conference, Uppsala, Sweden, August 24-28, 2009, Proceedings, Part I*. S.l.: Springer, Berlin, Heidelberg. pp. 356–359.

SANTOS, R.A., GÓES, V.A., ALMEIDA, L.F. DE, 2012. "Metodologia OriGame : um processo de desenvolvimento de jogos". In: *XI Brazilian Symposium on Games and Digital Entertainment*. S.l.: s.n. 2012. pp. 125–131.

SCHELL, J., 2008, *The Art of Game Design: A Book of Lenses*. . San Francisco, CA, USA, Morgan Kaufmann Publishers Inc. Morgan Kaufmann.

SOKOLOWSKI, J.A., BANKS, C.M., 2008, *Principles of Modeling and Simulation: A Multidisciplinary Approach*. . S.l., John Wiley & Sons, Inc.

SONG, S., KIM, M., 2012, "Five Models of Players' Rule Behavior for Game Balance". In: *Cyberpsychology, Behavior, and Social Networking*. v. 15, pp. 498–502.

STENSTRÖM, C.D., BJÖRK, S., 2013. "Understanding Computer Role-Playing Games: A Genre Analysis Based on Gameplay Features in Combat Systems". In: *Workshop on Design Patterns in Games at Foundations of Digital Games 2013*. Crete, Greece: s.n. 2013.

STERMAN, J.D., 2000, *Business dynamics: Systems thinking and modeling for a complex world*. . S.l., Irwin McGraw-Hill.

SWEETSER, P., WYETH, P., 2005, "GameFlow: A Model for Evaluating Player Enjoyment in Games". In: *Computers in Entertainment*. v. 3, pp. 3.

SYRIANI, E., VANGHELUWE, H., 2008. "Programmed graph rewriting with time for simulation-based design". In: VALLECILLO, Antonio, GRAY, Jeff & PIERANTONIO, Alfonso (eds.), *Theory and Practice of Model Transformations*. Berlin, Heidelberg: Springer Berlin Heidelberg. Lecture Notes in Computer Science. pp. 91–106.

SYSTEM DYNAMICS SOCIETY, 2015. Disponível em: <http://www.systemdynamics.org/what-is-s/>. Acessado em: 27 Abril 2015.

SYUFAGI, M.A., HARIADI, M., PURNOMO, M.H., 2013, "Petri Net Model for Serious Games Based on Motivation Behavior Classification". In: *International Journal of Computer Games Technology*. v. 2013, pp. 1–12.

TAYLOR, A., 2018. Disponível em: <http://www.evilresource.com/resident-evil-2/maps/police-station>. Acessado em: 3 Fevereiro 2018.

TGI GROUP, 2015. Disponível em: <http://www.informatik.uni-hamburg.de/TGI/PetriNets/>. Acessado em: 20 Maio 2015.

THIELSCHER, M., 2010, "A general game description language for incomplete information games". In: *Proceedings of the Twenty-Fourth AAAI Conference on Artificial Intelligence (AAAI-10)*. pp. 994–999.

THIELSCHER, M., 2011, "GDL-II". In: *KI - Künstliche Intelligenz*. v. 25, pp. 63–66.

ULLMAN, J.D., 1998, *Elements of ML programming (ML97 ed.)*. . 2. S.l., Prentice-Hall, Inc.

VENNERS, B., 2003. Disponível em: <http://www.artima.com/intv/pythonP.html>. Acessado em: 11 Janeiro 2018.

VGMAPS, 2013. Disponível em: <http://vgmaps.com/Atlas/NES/SuperMarioBros3-World1.png>. Acessado em: 5 Fevereiro 2018.

VOGEL, J.J., VOGEL, D.S., CANNON-BOWERS, J., et al., 2006, "Computer Gaming and Interactive Simulations for Learning: A Meta-Analysis". In: *Journal of Educational Computing Research*. v. 34, pp. 229–243.

WANG, J., 2007. "Petri Nets for Dynamic Event-Driven System Modeling". In: FISHWICK, Paul (ed.), *Handbook of Dynamic System Modeling*. S.l.: CRC Press. 2007. pp. 1–17.

WEBER, M., KINDLER, E., 2003, "The petri net kernel". In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*. v. 2472, pp. 109–124.

WESTERGAARD, M., KRISTENSEN, L.M., 2009. "The Access/CPN framework: A tool for interacting with the CPN tools simulator". In: WOLF, Karsten & FRANCESCHINIS, Giuliana (eds.), *Applications and Theory of Petri Nets*. Paris, France: Springer Berlin Heidelberg. pp. 313–322.

WESTERGAARD, M., LASSEN, K.B., 2006. "The BRITNeY suite animation tool". In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*. S.l.: s.n. 2006. pp. 431–440.

WINN, B.M., 2009. "The design, play, and experience framework". In: FERDIG, Richard E. (ed.), *Handbook of Research on Effective Electronic Gaming in Education*. Hershey: Information Science Reference (an imprint of IGI Global).

WIXON, D., 2006, "What is a game?". In: *interactions*. v. 13, pp. 37.

WOLSTENHOLME, E., 2005, "The potential of system dynamics". In: *Leading Edge*.

WORLD RPS SOCIETY, 2015. Disponível em: <http://worldrps.com/>. Acessado em: 15 Julho 2015.

XEXÉO, G., CARMO, A., ACIOLI, A., et al., 2017. *O Que São Jogos*. Rio de Janeiro. Disponível em: <http://www.cos.ufrj.br/uploadfile/publicacao/2766.pdf>.

Anexo I: Gênero de jogos

Gêneros de jogos pelo site Gamespot (www.gamespot.com), em 02/01/2018:

2D, 3D, 4X, Action, Adventure, Arcade, Baseball, Basketball, Beat-'Em-Up, Billiards, Bowling, Boxing, Card Game, Compilation, Cricket, Defense, Driving/Racing, Edutainment, Fighting, First-Person, Fitness, Fixed-Screen, Flight, Football (American), Free-to-Play, Gambling, Golf, Hidden Object, Hockey, Hunting/Fishing, Light-Gun, Management, Matching/Stacking, Miscellaneous, MMO, MOBA, Music/Rhythm, On-Rails, Open-World, Party/Minigame, Pinball, Platformer, Puzzle, Real-Time, Roguelike, Role-Playing, Scrolling, Shoot-'Em-Up, Shooter, Simulation, Skateboarding/Skating, Snowboarding/Skiing, Soccer, Sports, Strategy, Survival, Tactical, Team-Based, Tennis, Text-Based, Third-Person, Track' & Field, Trivia/Board Game, Turn-Based, Vehicular Combat, Wakeboarding/Surfing, Wrestling, VR.

Categorias de jogos pelo site Boardgamegeek (www.boardgamegeek.com), em 02/01/2018 (BOARDGAMEGEEK, 2018):

Abstract Strategy, Action / Dexterity, Adventure, Age of Reason, American Civil War, American Indian Wars, American Revolutionary War, American West, Ancient, Animals, Arabian, Aviation / Flight, Bluffing, Book, Card Game, Children's Game, City Building, Civil War, Civilization, Collectible Components, Comic Book /, Strip, Deduction, Dice, Economic, Educational, Electronic, Environmental, Expansion for Base-game, Exploration, Fan Expansion, Fantasy, Farming, Fighting, Game, System, Horror, Humor, Industry / Manufacturing, Korean War, Mafia, Math, Mature / Adult, Maze, Medical, Medieval, Memory, Miniatures, Modern Warfare, Movies / TV / Radio theme, Murder/Mystery, Music, Mythology, Napoleonic, Nautical, Negotiation, Novel-based, Number, Party Game, Pike and Shot, Pirates, Political, Post-Napoleonic, Prehistoric, Print & Play, Puzzle, Racing, Real-time, Religious, Renaissance, Science Fiction, Space Exploration, Spies/Secret Agents, Sports, Territory Building, Trains, Transportation, Travel, Trivia, Video Game Theme, Vietnam War, Wargame, Word Game, World War I, World War II, Zombies.

Anexo II: Experiência com Rede de Petri

Redes de Petri foram criadas em 1939 por Carl Adam Petri, quando na ocasião possuía treze anos. Seu trabalho foi documentado somente durante o seu doutorado, em 1962 (PETRI, 1962). Uma vez que é criado o modelo de Rede de Petri de um sistema, seja este existente ou proposto, ele pode ser simulado e analisado. A análise de uma Rede de Petri pode descobrir problemas ou detectar possibilidades de melhorias com o sistema modelado, contribuindo para a produção de um sistema melhor (PETERSON, 1978). Em outras palavras, pode ser utilizada para descrever e analisar sistemas interativos, como um videogame, e seu fluxo, por meio de simulação (BARRETO, 2013). A Figura 66 ilustra onde se encaixa uma Rede de Petri em seu uso no desenvolvimento de sistemas.

Os aspectos teóricos de Rede de Petri permitem modelagem e análise do comportamento de um sistema, enquanto a representação gráfica permite a visualização das mudanças de estado do sistema modelado.

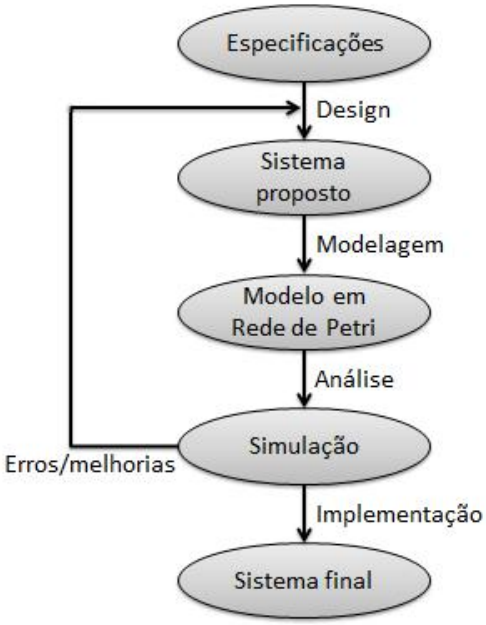

Figura 66: Exemplo de uso para Rede de Petri no *design*, análise e implementação de sistemas. Adaptado de (PETERSON, 1978)

Uma Rede de Petri é um tipo particular grafo direcionado bipartido, preenchido por três tipos de objetos:

- Posição ou lugar (*place*);
- Transição (*transition*);
- Arco direcionado (*arc*).

As posições ou lugares são nós circulares que, de forma abstrata, representam condições, atividades ou recursos. As transições representam eventos. Os arcos direcionados conectam posições a transições, ou transições a posições. A Rede de Petri mais simples que pode ser construída é representada por uma transição em conjunto com uma posição de entrada, uma posição de saída, e arcos direcionados conectando-os (WANG, 2007). A figura a seguir ilustra essa rede.

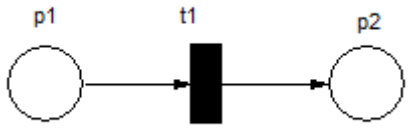

Figura 67: Rede de Petri mais simples

A posição a frente de um arco direcionado é chamada posição de entrada (*input place*) e, de maneira análoga, a posição na cauda de um arco é chamada posição de saída (*output place*). Os arcos são ponderados, o que significa que um certo número de *tokens* é necessário para que uma transição seja disparada. Os arcos sem peso são interpretados como arcos de peso unitários (PETERSON, 1977).

Para estudar o comportamento dinâmico de um sistema modelado com Redes de Petri, em termos de seu comportamento e mudanças de estado, cada posição pode conter zero ou um número positivo de *tokens*. A presença ou ausência de *token* na posição indica se uma condição associada com este lugar é verdadeira ou falsa.

Uma transição está pronta para ser disparada quando existem *tokens* suficientes em suas posições de entrada. Esta ação consome os *tokens* das posições de entrada e produz *tokens* nas posições de saída (WANG, 2007). É importante destacar que o disparo de uma transição é um evento atômico, isto é, o ato de consumo e produção de *tokens* sobre uma dada transição é considerado um passo, em termos de simulação.

Redes de Petri podem ser definidas formalmente como a tupla $N = (P, T, I, O, M_0)$, onde:

I. P é um conjunto finito de posições;

II. T é um conjunto finito de transições;

III. *I: P x T* é uma função de entrada que define os arcos direcionados de posições para transições;

IV. *O: T x P* é uma função de saída que define os arcos direcionados de transições para posições;

V. M_0 é a marcação inicial, isto é, a configuração inicial da quantidade de *tokens* em cada posição.

A figura a seguir apresenta um exemplo de uma rede simples, com três *tokens* em suas posições iniciais.

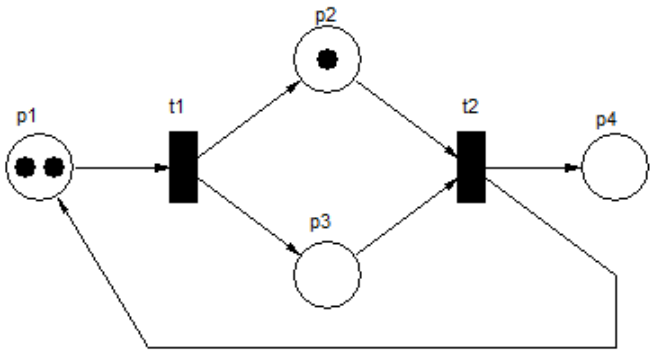

Figura 68: Exemplo de Rede de Petri em sua marcação inicial

A Rede de Petri apresentada na Figura 68 é matematicamente representada por:

$P = \{p1, p2, p3, p4\}$

$T = \{t1, t2\}$

$I =$

	t1	t2
p1	0	1
p2	1	0
p3	1	0
p4	0	1

$O =$

	t1	t2
p1	1	0
p2	0	1
p3	0	1
p4	0	0

$M_0 = (2 \quad 1 \quad 0 \quad 0)$

No exemplo apresentado, a marcação inicial indica que a posição *p1* é iniciada com dois *tokens*; a posição *p2* é iniciada com um *token*; e as demais posições se encontram inicialmente vazias. Todos os arcos são possuem peso unitário.

Em Redes de Petri, as transições representam eventos, ou seja, são responsáveis por modificar a marcação da rede. Para que o disparo de uma transição ocorra é necessário que existam *tokens* nas posições de entrada da transição. O exemplo da Figura 69 representa os estados de uma Rede de Petri, cujos arcos possuem pesos diferentes, antes e depois do disparo da transição t1.

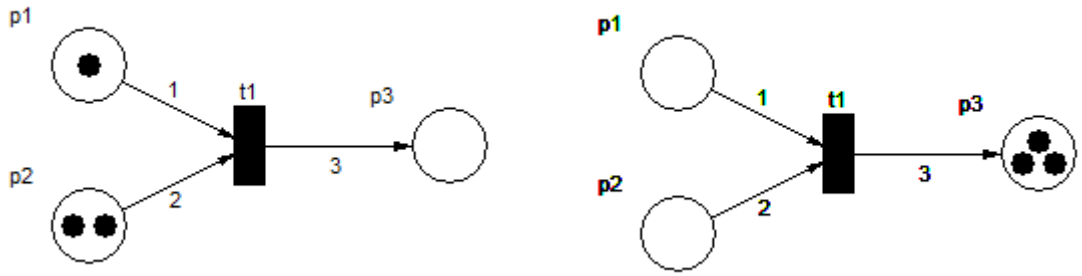

Figura 69: Marcação antes e depois do disparo da transição t1

O exemplo da Figura 70 demonstra uma Rede de Petri que não pode ser disparada, pois a posição de entrada p1 não contém a quantidade mínima de *tokens* estipulada pelo peso do arco para que a transição seja disparada.

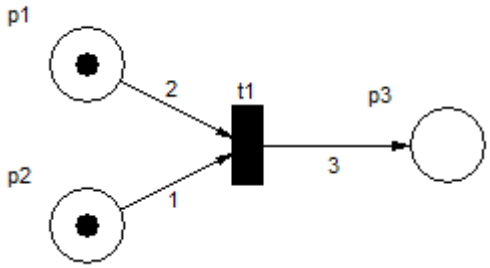

Figura 70: Exemplo de transição que não pode ser disparada

A Rede de Petri é executada através dos disparos das transições. A cada evento de disparo os *tokens* se movimentam pela rede modelada e a dinâmica do seu comportamento pode ser estudada.

Ao longo do tempo foram criadas extensões às Redes de Petri para melhorar o seu poder de representatividade, pois, originalmente, Redes de Petri não consideram aspectos

complexos como temporização, controle de fluxo e conceitos hierárquicos. A próxima seção fornece mais detalhes sobre essas extensões.

Redes de Petri de Alto-Nível e Redes de Petri Coloridas

Redes de Petri de Alto-Nível (RPAN) é a maneira como ficaram conhecidas as várias extensões que surgiram para estender o formalismo inicial das Redes de Petri. Existem ferramentas que suportam várias dessas extensões simultaneamente.

As Redes de Petri Temporizadas adicionam a variável tempo ao modelo, permitindo estipular tempo de atraso para o disparo de transições. As Redes de Petri Hierárquicas permitem subdividir a rede original em níveis, o que facilita seu projeto e compreensão. As Redes de Petri Coloridas são as mais famosas, por vezes sendo mencionadas na literatura como se possuíssem todas as particularidades das RPAN. Essa confusão pode ter sido motivada devido a uma das ferramentas mais famosas, *CPN Tools*, possuir esse nome que faz alusão a Redes de Petri Coloridas, mas dar suporte também a outros tipos de redes. A figura a seguir sumariza esse cenário.

Figura 71: Redes de Petri de Alto-Nível

Redes de Petri Colorida (ou CPN, *Coloured Petri Net*) é uma linguagem gráfica para a construção de modelos de sistemas concorrentes e análise de suas propriedades. CPN é uma linguagem de modelagem de eventos discretos que combina as capacidades de Redes de Petri com os recursos de uma linguagem de programação de alto nível (JENSEN & KRISTENSEN, 2009). A linguagem de programação CPN ML é baseada em *Standard ML*, uma linguagem de programação funcional que fornece primitivos para a definição de tipos de dados, para descrição de funções de manipulação de dados, e para a criação de modelos compactos e parametrizáveis (JENSEN & KRISTENSEN, 2009).

Nas Redes de Petri Coloridas os *tokens* passam a conter informações, isto é, dados de diferentes tipos, chamados de cores. As cores podem ser tipos básicos ou podem ser construídas a partir de uma combinação desses tipos. Cada posição armazena *tokens* de certo tipo, e os arcos realizam operações sobre os *tokens*. Os tipos básicos são: INT; STRING; BOOL; REAL; e UNIT. Os tipos UNIT são unitários e, por esse motivo, também chamados de sem cor (JENSEN, 1996).

Os tipos básicos podem ser combinados por meio dos construtores para criação de novas cores. Os construtores disponíveis são: WITH; PRODUCT; RECORD; LIST. O disparo das transições numa CPN pode ser condicionado com o uso de operadores básicos e lógicos.

A definição de cores permite uma criação bem detalhada dos *tokens* que serão utilizados no modelo. Para fins de exemplo, considere um modelo com a seguinte definição de cores:

```
colset Idade = INT WITH 0..130;
colset Humano = WITH homem | mulher | crianca;
colset HumanoIdade = PRODUCT Humano * Idade;
```

Os *tokens* do modelo devem ser restritos às cores definidas, de forma que a cor Idade pode conter *tokens* com valores inteiros entre 0 e 130; a cor Humano pode conter somente o valor “homem”, ou “mulher”, ou “criança”. E por fim, a cor HumanoIdade é um produto das cores definidas previamente. Possíveis valores para a cor HumanoIdade seriam (homem,30), (mulher,25), (crianca,7) etc.

A Figura 72 apresenta um exemplo de modelo simples de Rede de Petri Colorida construído com a ferramenta *CPN Tools*.

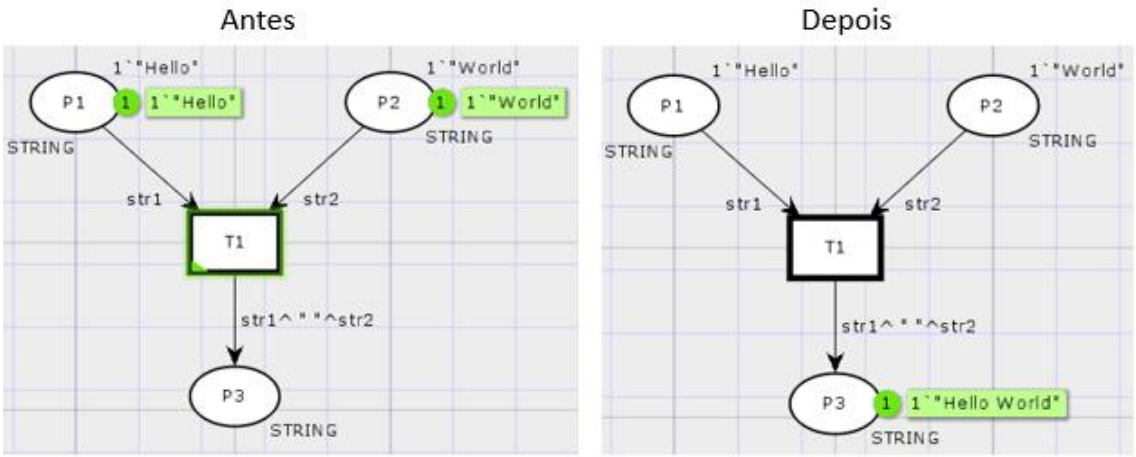

Figura 72: Exemplo de disparo de transição em Rede de Petri Colorida

As cores definidas para o exemplo da Figura 72 são:

```
colset STRING;
var str1, str2: STRING;
```

O exemplo apresentado possui as posições P1, P2 e P3 e uma transição T1. Todas as posições possuem a cor STRING. As posições P1 e P2 possuem, em sua marcação inicial, os *tokens* “Hello” e “World”, respectivamente.

Quando a transição T1 é disparada os *tokens* de P1 e P2 são consumidos. A variável str1 é preenchida com o valor “Hello”, a variável str2 é preenchida com o valor “World”, a posição P3 recebe um novo *token*, formado pela concatenação dos valores das variáveis str1 e str2. Após o disparo de T1, não há nenhuma outra transição a ser disparada, isto é, o modelo apresentado possui somente um passo para sua completa execução.

ML

ML é uma linguagem de programação funcional, imperativa, com o poder de funções matemáticas, criada por Robin Milner no início da década de 70 na *University of Edinburgh*. ML, que é derivado de *meta linguagem*, é fortemente tipado e foi a primeira linguagem de programação a fornecer uma forma particular de tipo polimórfico que torna a tipagem forte notavelmente flexível (MILNER *et al.*, 1997).

ML é muitas vezes referida como uma linguagem funcional impura, porque permite efeitos secundários, algo incomum em linguagens de programação puramente funcionais. A função é uma característica fundamental em UML, os programas escritos nessa linguagem não possuem instruções ou comandos, mas sim expressões que são avaliadas (ULLMAN, 1998).

ML serve de base a diversas outras linguagens de programação funcional, sendo *Caml* e *Standard ML* os dois maiores dialetos. Este último, conforme mencionado previamente, é a linguagem-base da CPN ML.

A Ferramenta *CPN Tools*

Existem diversas ferramentas que suportam RPAN. O *website* Petri Nets World (TGI GROUP, 2015) da *University of Hamburg* é supervisionado por um comitê formado por importantes pesquisadores da área. O *website* possui uma extensa relação de

ferramentas de Redes de Petri e apresenta detalhes, para cada ferramenta, dos tipos de Redes de Petri suportados, seus principais componentes e plataforma operacional.

CPN Tools é uma ferramenta para edição, simulação e análise de Redes de Petri Coloridas, Hierárquicas, com suporte a temporização (RATZER *et al.*, 2003). É o resultado de um projeto de pesquisa nomeado CPN2000 (CPN 2000 PROJECT, 2003) (JENSEN, 2015), cujo objetivo era usufruir dos desenvolvimentos na área de interação humano-computador e realizar uma reformulação completa da interface gráfica do *Design/CPN* (DESIGN/CPN, 2004).

Na ferramenta *CPN Tools*, as inscrições e declarações das posições, transições e arcos são feitas utilizando a linguagem CPN ML. Essa linguagem permite a extensão de tipos e um armazenamento automático eficiente para estrutura de dados e funções, além de facilitar a programação com estrutura de dados recursiva e simbólica. A figura a seguir apresenta a interface do *CPN Tools*, em sua versão 4.0.

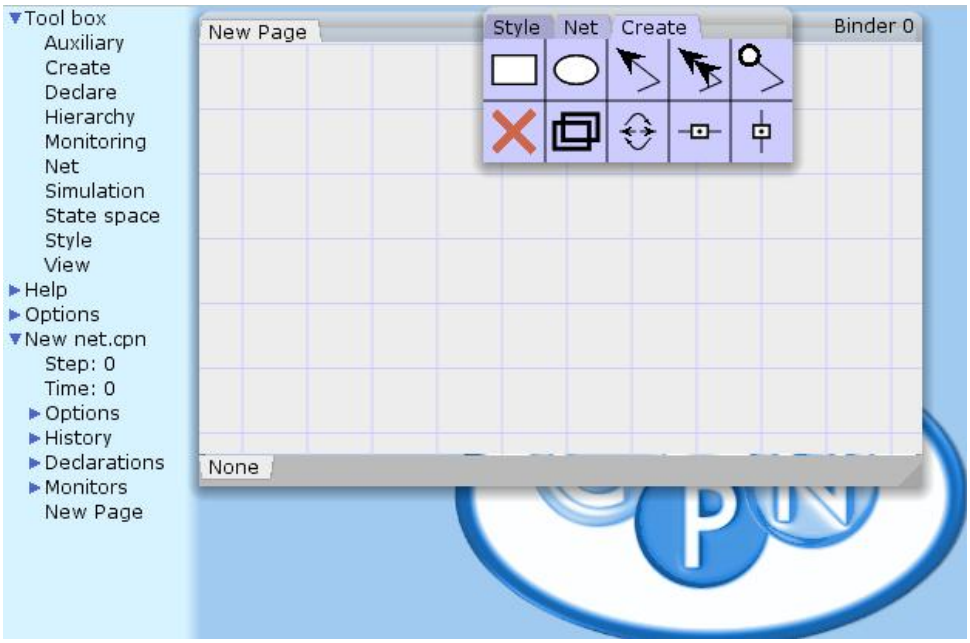

Figura 73: Interface da ferramenta *CPN Tools*

Apesar do nome fazer uma clara referência a Redes de Petri Coloridas, *CPN Tools* possui suporte a RPAN, sendo a ferramenta mais utilizada. Sua versão mais recente, até o momento, é a 4.0.1 lançada em fevereiro de 2015 e executa sobre o ambiente Windows. Para Mac/Linux a versão mais recente é a 2.3.5 (EINDHOVEN UNIVERSITY OF TECHNOLOGY, 2015). Além da figura da interface apresentada, o exemplo da Figura 72 foi construído utilizando a ferramenta.

A interface interativa do *CPN Tools* permite utilização completa dos recursos de Redes de Petri de Alto-Nível, tais como criação de *tokens* valorados, definição de variáveis e tipos, definições de tempos para disparo de transição, simulação passo a passo, etc. A ferramenta indica erros contextuais e relações de dependência entre os elementos da rede, fornece ferramentas de análise e simulação de redes.

Jogos e Redes de Petri

Em (BROM & ABONYI, 2006) e (DANIEL BALAS, CYRIL BROM, ADAM ABONYI, 2008) é desenvolvido uma técnica para a representação de enredo não-linear de jogo e seu gerenciamento. A técnica visa aplicação em jogos com mundos extensos povoados por personagens virtuais, como os *Massively Multiplayer Online Role-Playing Game* (MMORPG). As Redes de Petri são utilizadas para controlar interativamente o plano de história do jogo, baseado nas ações individuais de cada jogador.

Em (LIU & WU, 2013), inspirados pelas partidas de basquete da NBA, que por muitas vezes são decididas em seus últimos segundos, os autores criaram uma Rede de Petri para representar as possíveis jogadas de ataque que podem ser realizadas nesses últimos segundos decisivos de jogo. O objetivo dos autores é analisar a Rede criada e extrair informações que possam auxiliar na tomada de decisões em relação a escolha de jogadas que podem ser executadas com mais chances de levar o time a vitória.

Redes de Petri também já foram utilizadas para auxiliar na medição da motivação gerada por um jogo. Em (SYUFAGI *et al.*, 2013) uma Rede de Petri foi modelada para classificar a motivação dos jogadores enquanto estes interagem com jogos sérios. O comportamento dos jogadores e suas escolhas são utilizados pela Rede de Petri para determinar o nível de motivação do jogador e classificá-lo dentre uma das três categorias criadas pelos autores: esforço mental (*mental effort*); persistência (*persistence*); e escolha ativa (*active choice*).

Em (GRÜNVOGEL *et al.*, 2004) é apresentada uma maneira de representar cenários de missões de videogames, com modelagem de relações espaço-temporais. As operações lógicas e temporais são modeladas usando Redes de Petri, e relações topológicas do universo do jogo por hipergrafos. O jogador é o principal recurso do jogo na Rede de Petri e suas ações estão associadas às posições. No estudo de caso realizado, o modelo é utilizado para representar todas as possíveis sequências de ações que o jogador precisa executar para completar uma determinada fase do jogo. Algumas ações podem

ser executadas fora de ordem, algumas ações só podem ser executadas se outras forem completadas anteriormente. Com o modelo construído, é possível realizar simulações para entender os possíveis comportamentos do jogador.

A partir do entendimento de que missões de videogames são similares a processos de *workflow,* Oliveira em (DE OLIVEIRA *et al.*, 2011) refinou o trabalho de Grünvogel, propondo que a Rede de Petri que representa um cenário de jogo seja uma *WorkFlow* net e obedeça ao critério *Soundness*, isto é, um tipo particular de rede que satisfaz os seguintes requisitos:

- Para cada *token* na posição inicial, um e somente um aparece na posição final;
- Quando o *token* aparece na posição final, todas as outras posições estão vazias;
- Para cada transição, é possível mover do estado inicial a um estado em que a transição está disponível, i.e. não existem transições mortas.

A lógica linear é utilizada como prova para verificar se a rede modelada cumpre os critérios estabelecidos. Assim como em (GRÜNVOGEL *et al.*, 2004), as transições e posições da Rede de Petri são baseadas nas ações disponíveis para o jogador em cada etapa. Não há representação de aspectos dos recursos ou da economia do jogo.

Araújo e Roque utilizaram Redes de Petri para modelagem do *design* de jogo e seu fluxo. Em (ARAÚJO & ROQUE, 2009) os autores descrevem dois estudos de caso e analisam que a representação de jogo com Redes de Petri pode ser de grande utilidade para os *designers*, pois a representação permite identificar antecipadamente problemas como, por exemplo, balanceamento do fluxo de jogo, uma tarefa geralmente realizada em estágios mais avançados do processo de criação de jogo, quando já se possui um protótipo jogável. No entanto, à medida que a complexidade do jogo aumenta, também aumenta a dificuldade da modelagem, de forma que a fidelidade da representação é comprometida.

Kriglstein iniciou um estudo utilizando padrões de *workflow* para modelar a sucessão de tarefas em jogos (KRIGLSTEIN *et al.*, 2014). Trata-se ainda de um estudo preliminar com uso de BPMN 2.0 na representação dos modelos, mas que, segundo os autores, pretende no futuro utilizar o formalismo de Redes de Petri na representação dos padrões de *workflow*.

Anexo III: Artefatos de simulações

Simulação do jogo de aventura

Os personagens, mundo de jogo e itens foram construídos utilizando os componentes auxiliares de fabricação destes artefatos de entrada. O arquivo XML fornecido para o simulador se encontra a seguir.

```
<root>
 <world>
  <place id='1' neighbors='2'>
   <foe name='Goblin' lif='25' atk='6' dfs='2' agi='10' quantity='2'/>
  </place>
  <place id='2' neighbors='3,4'>
   <item id='Light Sword' type='atk' value='5' quantity='1' />
   <foe name='Orc' lif='50' atk='13' dfs='6' agi='11'/>
  </place>
  <place id='3' neighbors='5'>
   <foe name='Goblin' lif='25' atk='6' dfs='2' agi='10' quantity='2'/>
   <foe name='Orc' lif='50' atk='13' dfs='6' agi='11'/>
  </place>
  <place id='4' neighbors='6'>
   <foe name='Goblin' lif='25' atk='6' dfs='2' agi='10' quantity='2'/>
   <foe name='Troll' lif='80' atk='14' dfs='8' agi='4'/>
  </place>
  <place id='5' neighbors='7'>
   <item id='shield' type='dfs' value='10' quantity='1' />
   <item id='silverKey' type='key'/>
   <foe name='Goblin' lif='25' atk='6' dfs='2' agi='10'/>
   <foe name='Troll' lif='80' atk='14' dfs='8' agi='4' quantity='2'/>
  </place>
  <place id='6' neighbors='7'>
   <item id='Great Sword' type='atk' value='10' quantity='1' />
   <foe name='Goblin' lif='25' atk='6' dfs='2' agi='10'/>
   <foe name='Orc' lif='50' atk='13' dfs='6' agi='11' quantity='2'/>
  </place>
  <place id='7' neighbors='8,9'>
   <item id='potion' type='lif' value='80' quantity='1' />
   <foe name='Orc' lif='50' atk='13' dfs='6' agi='11'/>
   <foe name='Troll' lif='80' atk='14' dfs='8' agi='4'/>
  </place>
  <place id='8' neighbors='12'>
   <foe name='Goblin' lif='25' atk='6' dfs='2' agi='10' quantity='8'/>
  </place>
  <place id='9' neighbors='10,11'>
   <foe name='Goblin' lif='25' atk='6' dfs='2' agi='10' quantity='2'/>
   <foe name='Orc' lif='50' atk='13' dfs='6' agi='11'/>
   <foe name='Troll' lif='80' atk='14' dfs='8' agi='4'/>
  </place>
```

```
    <place id='10' neighbors='11' idLock='silverKey'>
      <item id='potion' type='lif' value='80' quantity='1' />
      <item id='Medium Sword' type='atk' value='8' quantity='1' />
      <foe name='Troll' lif='80' atk='14' dfs='8' agi='4' quantity='2'/>
    </place>
    <place id='11' neighbors='12'>
      <foe name='Goblin' lif='25' atk='6' dfs='2' agi='10' quantity='4'/>
    </place>
    <place id='12' final='true'>
      <foe name='Great Warlock' lif='200' atk='18' dfs='9' agi='5'/>
    </place>
  </world>
</root>
```

Os relatórios resumido e detalhado possuem 367.894 e 2.649.275 linhas, respectivamente. Sua síntese está abaixo.

```
----------Simulation report after 10000 expeditions:----------
Heroes successes: 4885 (48.85%)
Heroes failures: 5115 (51.15%)
Avg visited places: 7.4821
Avg items found: 3.7183
Turns avg per expedition: 177.0
Turns avg per places: 23.62
Most common visit order: 1, 2, 3, 5, 7, 8, 12. Per 2451 times (24.51%)
```

Os gráficos gerados são reproduzidos a seguir.

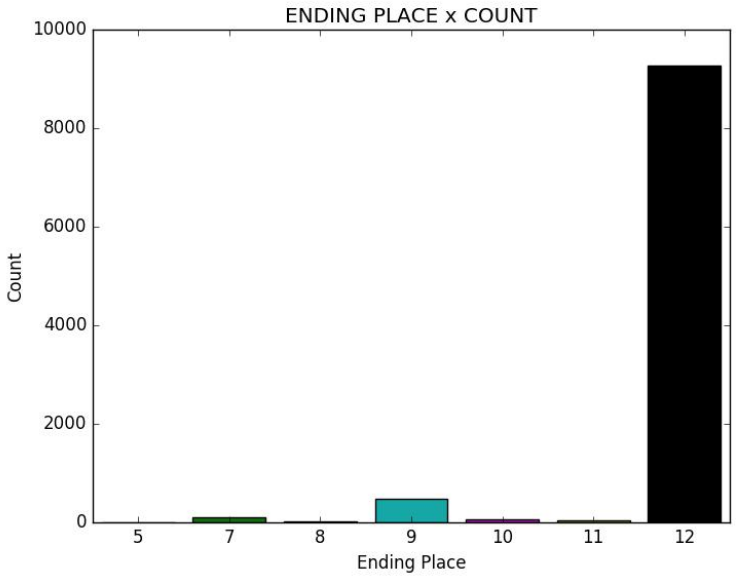

Figura 74: Local fim da simulação

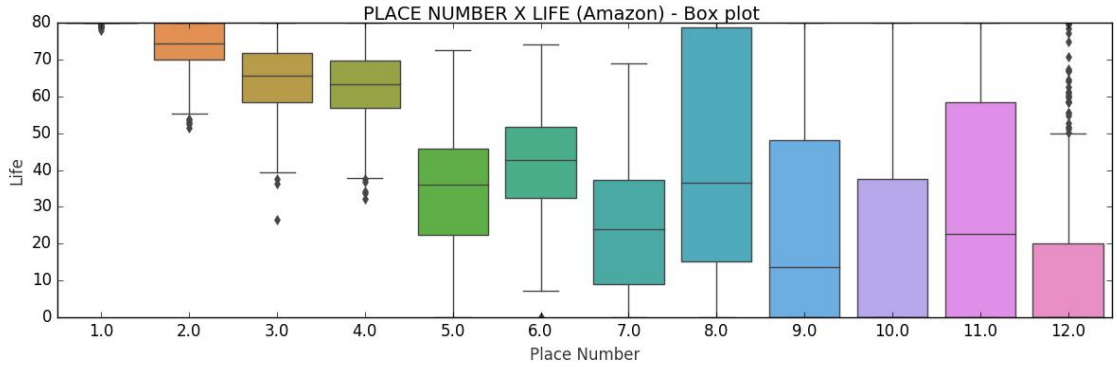

Figura 75: *Box plot* de variação do atributo vida do personagem *Amazon*

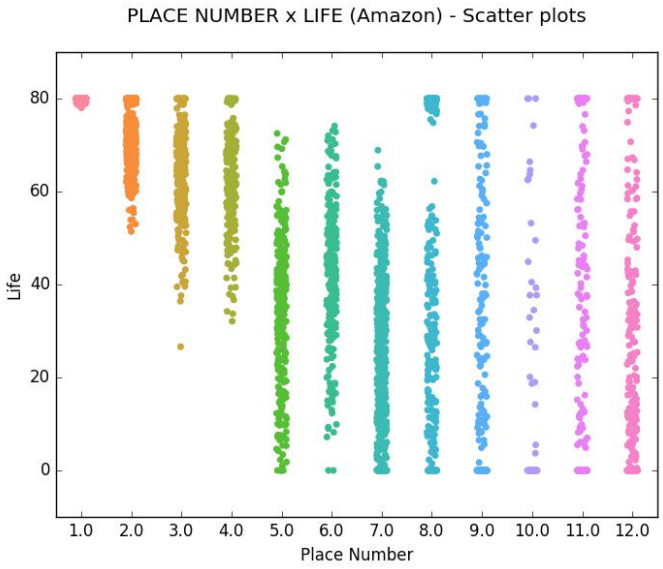

Figura 76: Dispersão do atributo vida do personagem *Amazon*

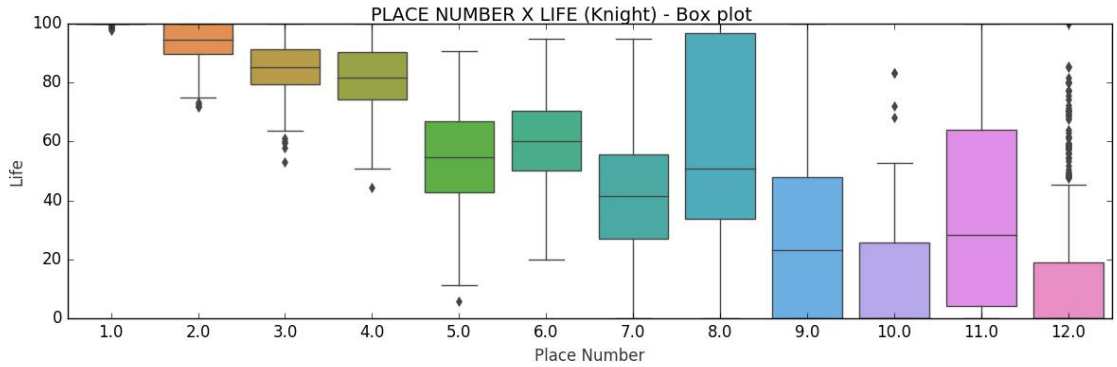

Figura 77: *Box plot* de variação do atributo vida do personagem *Knight*

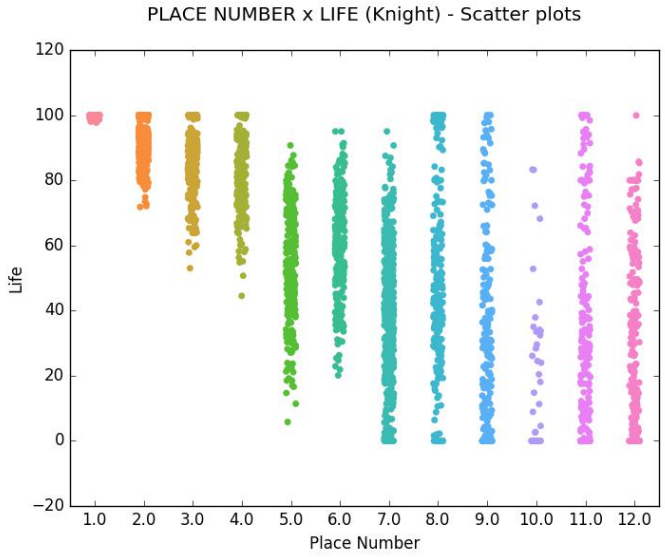

Figura 78: Dispersão do atributo vida do personagem *Knight*

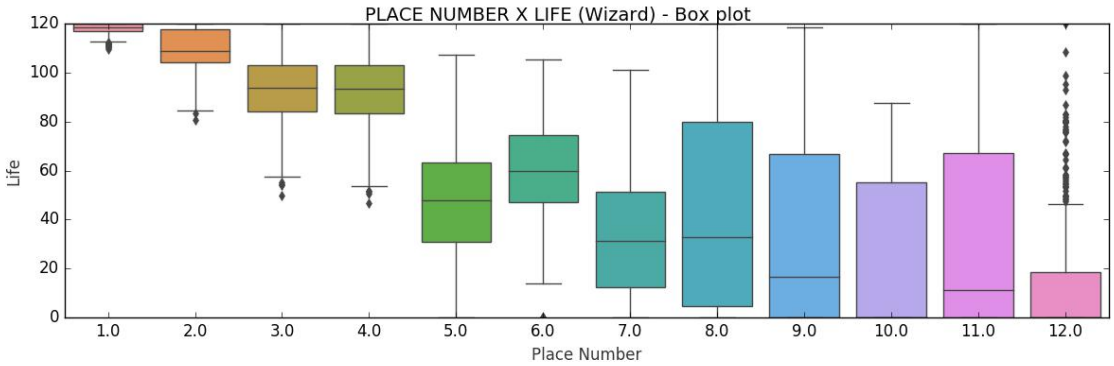

Figura 79: *Box plot* de variação do atributo vida do personagem *Wizard*

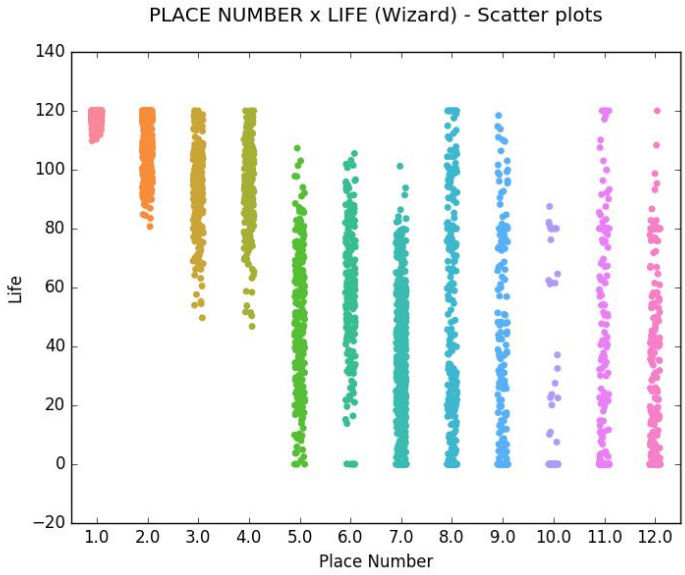

Figura 80: Dispersão do atributo vida do personagem *Wizard*

Figura 81: Média do atributo vida dos personagens

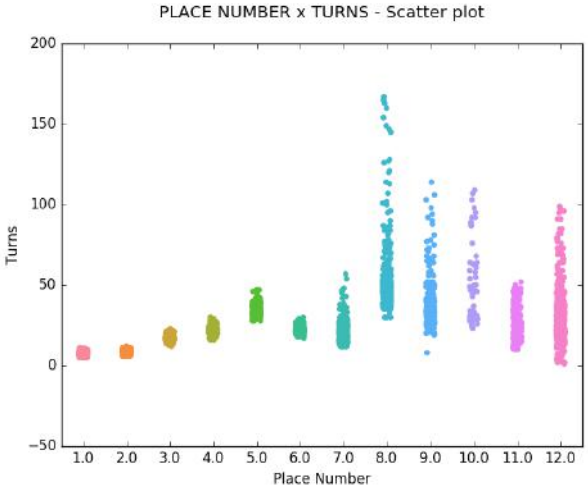

Figura 82: Dispersão da quantidade de turnos por local

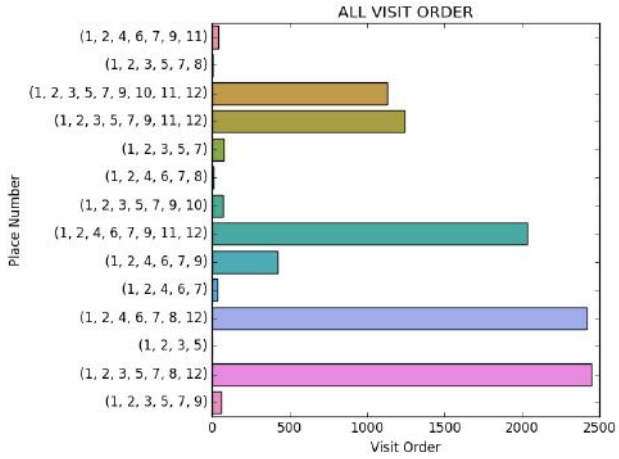

Figura 83: Ordem de visita aos locais

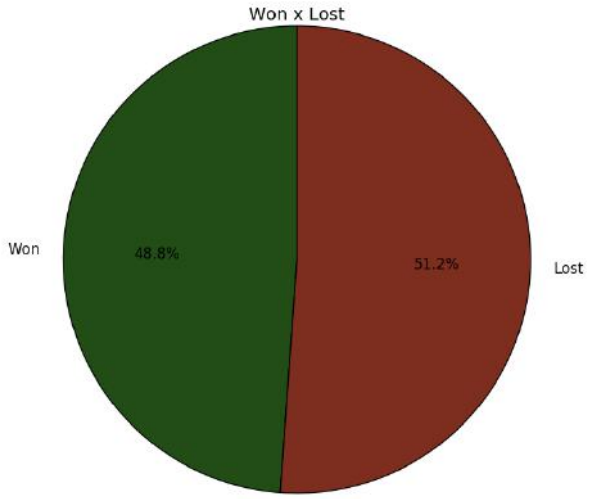

Figura 84: Sucesso e falha

Após algumas modificações foi executado nova bateria de simulações. O arquivo XML modificado e fornecido para o simulador se encontra a seguir.

```
<root>
 <world>
  <place id='1' neighbors='2'>
   <foe name='Goblin' lif='25' atk='6' dfs='2' agi='10' quantity='6'/>
  </place>
  <place id='2' neighbors='3,4'>
   <item id='Light Sword' type='atk' value='5' quantity='1' />
   <foe name='Goblin' lif='25' atk='6' dfs='2' agi='10'/>
   <foe name='Orc' lif='50' atk='13' dfs='6' agi='11' quantity=’2’/>
  </place>
  <place id='3' neighbors='5'>
   <foe name='Goblin' lif='25' atk='6' dfs='2' agi='10' quantity='2'/>
   <foe name='Orc' lif='50' atk='13' dfs='6' agi='11'/>
  </place>
  <place id='4' neighbors='6'>
   <foe name='Goblin' lif='25' atk='6' dfs='2' agi='10' quantity='2'/>
   <foe name='Troll' lif='80' atk='14' dfs='8' agi='4'/>
  </place>
  <place id='5' neighbors='7'>
   <item id='shield' type='dfs' value='10' quantity='1' />
   <item id='silverKey' type='key'/>
   <foe name='Goblin' lif='25' atk='6' dfs='2' agi='10'/>
   <foe name='Troll' lif='80' atk='14' dfs='8' agi='4' quantity='2'/>
  </place>
  <place id='6' neighbors='7'>
   <item id='Great Sword' type='atk' value='10' quantity='1' />
   <foe name='Goblin' lif='25' atk='6' dfs='2' agi='10'/>
   <foe name='Orc' lif='50' atk='13' dfs='6' agi='11' quantity='2'/>
  </place>
  <place id='7' neighbors='8,9'>
```

```
    <item id='potion' type='lif' value='80' quantity='1' />
    <foe name='Orc' lif='50' atk='13' dfs='6' agi='11'/>
    <foe name='Troll' lif='80' atk='14' dfs='8' agi='4'/>
   </place>
   <place id='8' neighbors='12'>
    <foe name='Goblin' lif='25' atk='6' dfs='2' agi='10' quantity='8'/>
    <foe name='Troll' lif='80' atk='14' dfs='8' agi='4'/>
   </place>
   <place id='9' neighbors='10,11'>
    <foe name='Goblin' lif='25' atk='6' dfs='2' agi='10' quantity='2'/>
    <foe name='Orc' lif='50' atk='13' dfs='6' agi='11'/>
    <foe name='Troll' lif='80' atk='14' dfs='8' agi='4'/>
   </place>
   <place id='10' neighbors='11' idLock='silverKey'>
    <item id='potion' type='lif' value='80' quantity='1' />
    <item id='Medium Sword' type='atk' value='8' quantity='1' />
    <foe name='Troll' lif='80' atk='14' dfs='8' agi='4' quantity='2'/>
   </place>
   <place id='11' neighbors='12'>
    <foe name='Goblin' lif='25' atk='6' dfs='2' agi='10' quantity='4'/>
   </place>
   <place id='12' final='true'>
    <foe name='Great Warlock' lif='200' atk='18' dfs='9' agi='5'/>
   </place>
  </world>
</root>
```

Os relatórios resumido e detalhado possuem 308.086 e 2.628.492 linhas, respectivamente. Sua síntese está abaixo.

```
----------Simulation report after 10000 expeditions:----------
Heroes successes: 433 (4.33%)
Heroes failures: 9567 (95.67%)
Avg visited places: 5.7784
Avg items found: 2.7935
Turns avg per expedition: 179.94
Turns avg per places: 30.45
Most common visit order: 1, 2, 4, 6, 7. Per 2310 times (23.1%)
```

Os gráficos gerados se encontram a seguir.

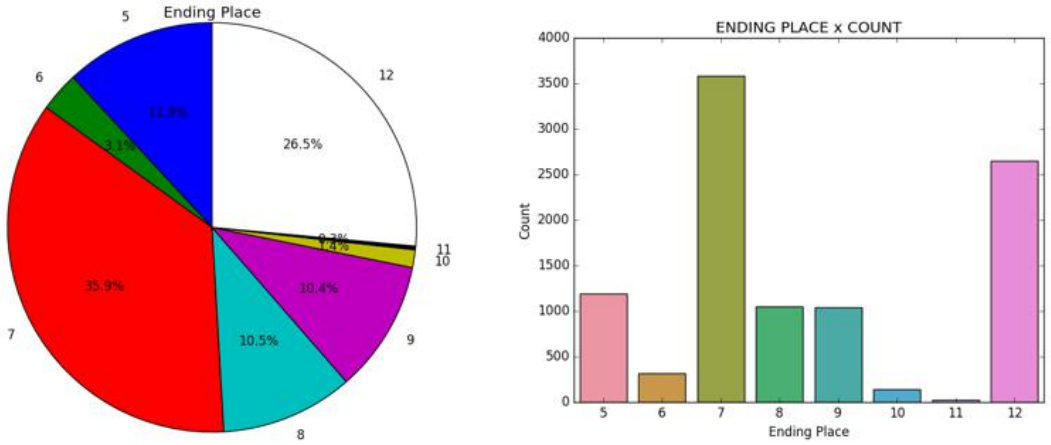

Figura 85: Local fim da simulação

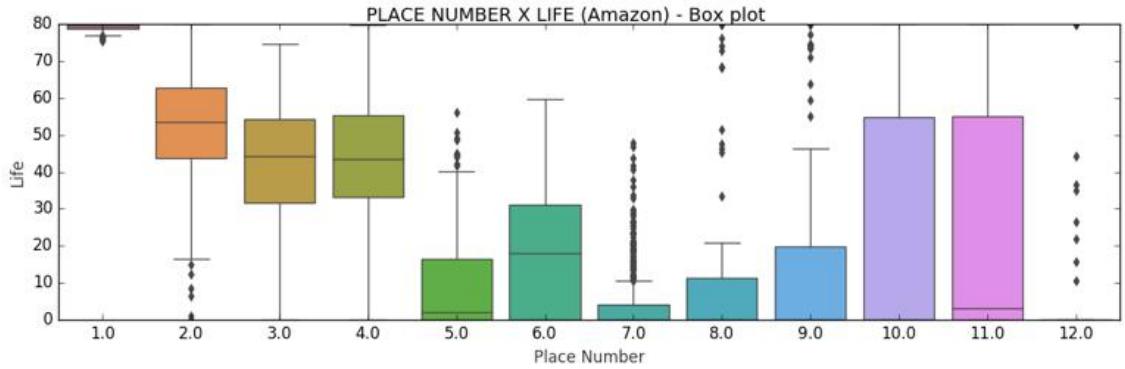

Figura 86: *Box plot* de variação do atributo vida do personagem *Amazon*

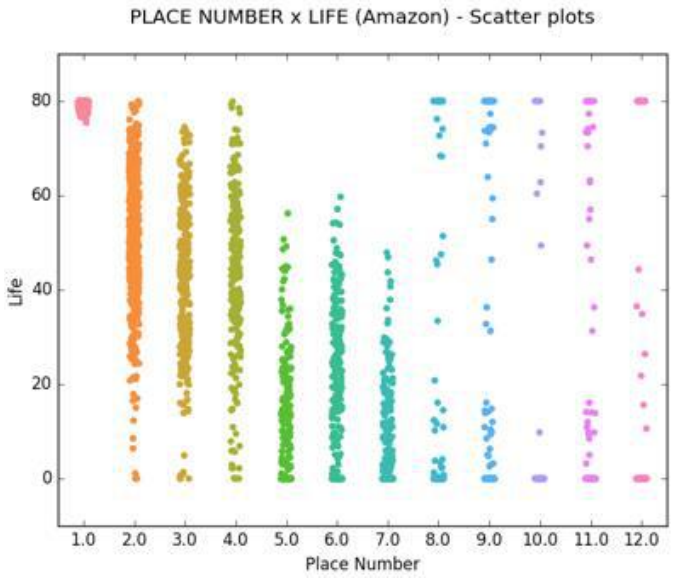

Figura 87: Dispersão do atributo vida do personagem *Amazon*

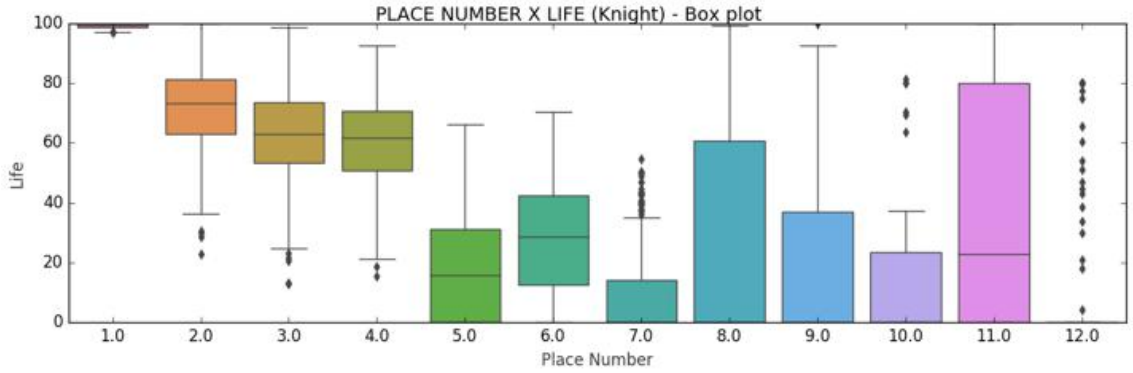

Figura 88: *Box plot* de variação do atributo vida do personagem *Knight*

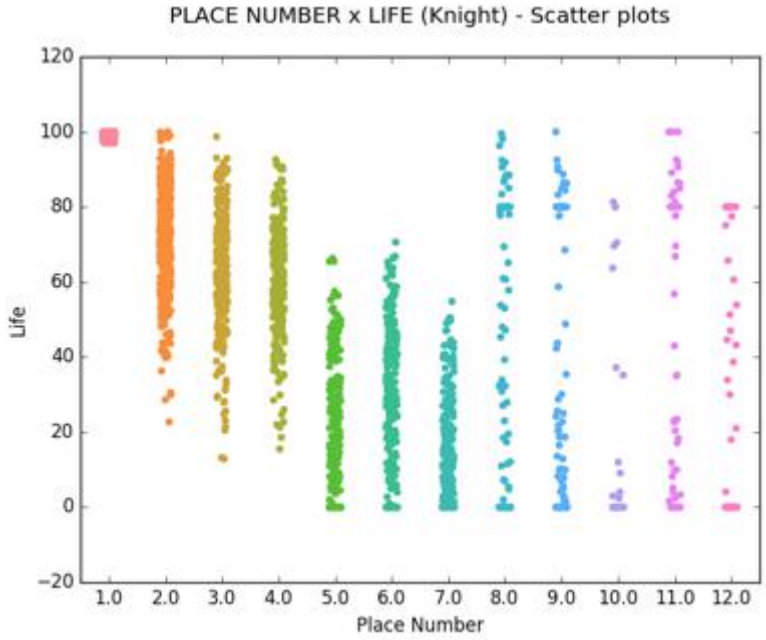

Figura 89: Dispersão do atributo vida do personagem *Knight*

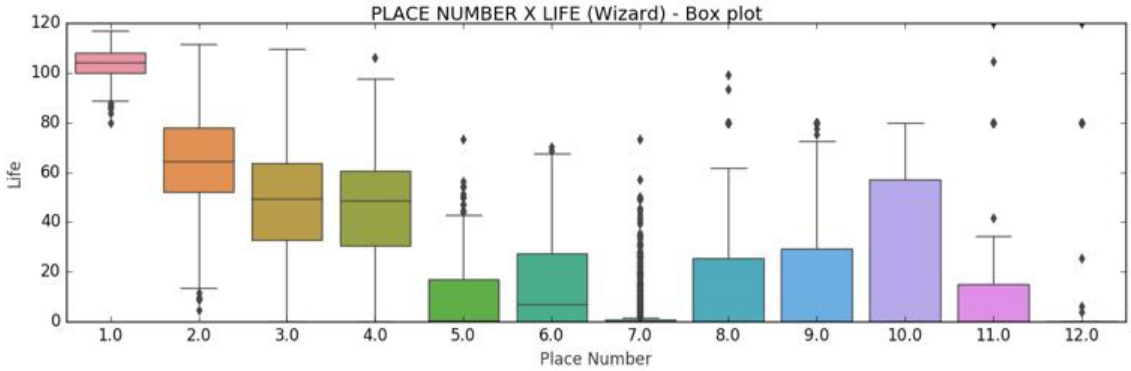

Figura 90: *Box plot* de variação do atributo vida do personagem *Wizard*

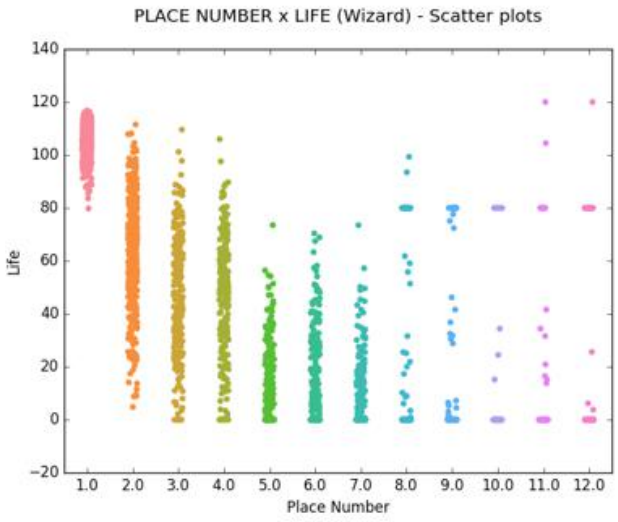

Figura 91: Dispersão do atributo vida do personagem *Wizard*

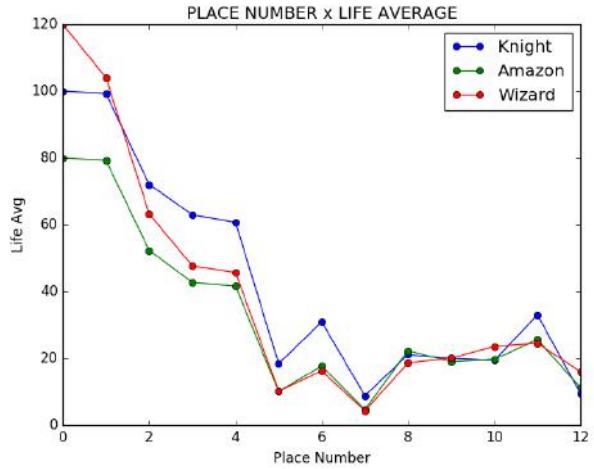

Figura 92: Média do atributo vida dos personagens

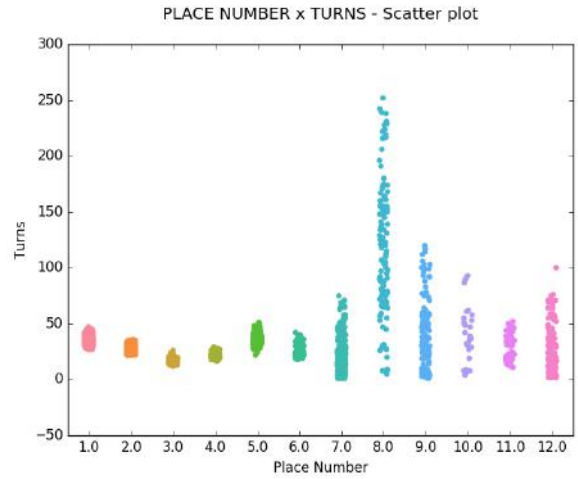

Figura 93: Dispersão da quantidade de turnos por local

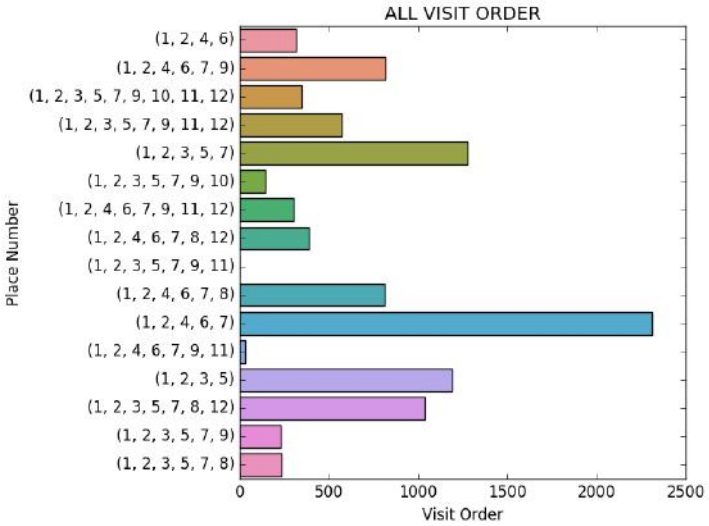

Figura 94: Ordem de visita aos locais

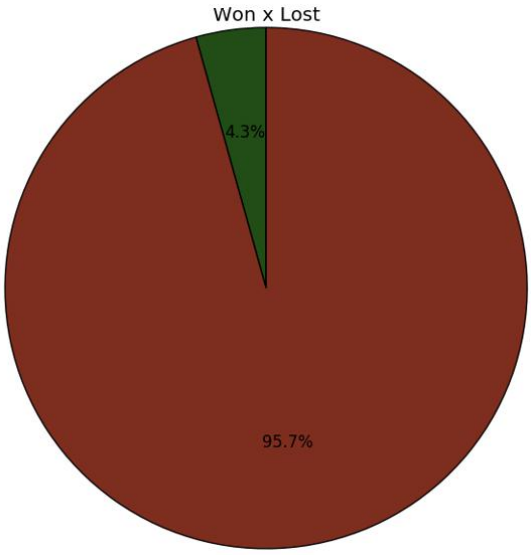

Figura 95: Sucesso e falha

Simulação do jogo com exploração

O arquivo XML a seguir corresponde ao mapa da Figura 59, representando o mapa do primeiro e segundo andar delegacia de polícia de *Resident Evil 2*.

```
<root>
 <world>
  <place id='1' neighbors='2,3,11,20'>
   <item id='blackKey' type='key'/>
  </place>
  <place id='2' neighbors='1,4' idLock='blueKey'/>
  <place id='3' neighbors='1,8' idLock='blackKey'>
   <item id='blueKey' type='key'/>
```

```
  </place>
  <place id='4' neighbors='2,5,6'/>
  <place id='5' neighbors='4'>
   <item id='greyKey' type='key'/>
  </place>
  <place id='6' neighbors='4,7,9'/>
  <place id='7' neighbors='6'>
   <item id='yellowKey' type='key'/>
  </place>
  <place id='8' neighbors='3,9'/>
  <place id='9' neighbors='6,8,10,25'/>
  <place id='10' neighbors='9'/>
  <place id='11' neighbors='1,12,18' idLock='redKey'/>
  <place id='12' neighbors='11,13,16'/>
  <place id='13' neighbors='12,34'/>
  <place id='14' neighbors='18'/>
  <place id='15' neighbors='18'/>
  <place id='16' neighbors='12,17,35' idLock='purpleKey'/>
  <place id='17' neighbors='16'/>
  <place id='18' neighbors='11,14,15,19'/>
  <place id='19' neighbors='18'/>
  <place id='20' neighbors='1,21,26'/>
  <place id='21' neighbors='20,22'>
   <item id='redKey' type='key'/>
  </place>
  <place id='22' neighbors='21,23' idLock='greenKey'/>
  <place id='23' neighbors='22,24,25'/>
  <place id='24' neighbors='23'>
   <item id='greenKey' type='key'/>
  </place>
  <place id='25' neighbors='9,23'>
   <item id='aquaKey' type='key'/>
  </place>
  <place id='26' neighbors='20,27'/>
  <place id='27' neighbors='26,28,29,32'/>
  <place id='28' neighbors='27' idLock='aquaKey'>
   <item id='purpleKey' type='key'/>
  </place>
  <place id='29' neighbors='27,30'/>
  <place id='30' neighbors='29,31'/>
  <place id='31' neighbors='30'/>
  <place id='32' neighbors='27,33,34' idLock='yellowKey'/>
  <place id='33' neighbors='32'/>
  <place id='34' neighbors='32,13' idLock='greyKey'/>
  <place id='35' final='true'/>
 </world>
</root>
```

Os relatórios resumido e detalhado possuem respectivamente 529.940 e 1.018.570 linhas, para mil simulações. Sua síntese se encontra a seguir.

----------Simulation report after 1000 expeditions:----------
Heroes successes: 1000 (100.0%)
Heroes failures: 0 (0.0%)
Avg visited places: 248.18
Avg items found: 7.73
Turns avg per expedition: 0.0
Turns avg per places: 0.0
Most common visit order: 1, 20, 21, 20, 26, 27, 29, 30, 31, 30, 31, 30, 29, 30, 31, 30, 31, 30, 31, 30, 31, 30, 31, 30, 29, 30, 31, 30, 31, 30, 29, 30, 29, 30, 31, 30, 31, 30, 29, 27, 29, 30, 31, 30, 31, 30, 29, 30, 31, 30, 29, 27, 26, 20, 26, 20, 1, 11, 18, 14, 18, 15, 18, 19, 18, 15, 18, 19, 18, 15, 18, 14, 18, 15, 18, 15, 18, 15, 18, 15, 18, 11, 12, 13, 12, 11, 18, 19, 18, 14, 18, 15, 18, 11, 1, 3, 8, 9, 10, 9, 25, 23, 24, 23, 22, 21, 22, 23, 22, 21, 20, 26, 20, 26, 27, 28, 27, 26, 20, 26, 20, 21, 22, 23, 22, 21, 22, 23, 25, 23, 25, 9, 6, 7, 6, 4, 5, 4, 2, 4, 2, 4, 5, 4, 5, 4, 2, 4, 6, 9, 6, 9, 25, 23, 24, 23, 25, 23, 25, 9, 25, 23, 24, 23, 25, 23, 25, 9, 6, 4, 2, 4, 6, 9, 8, 3, 8, 9, 6, 4, 2, 4, 5, 4, 2, 4, 2, 1, 20, 21, 20, 1, 11, 1, 3, 8, 9, 25, 9, 8, 9, 10, 9, 25, 9, 25, 23, 22, 23, 22, 23, 24, 23, 22, 21, 22, 21, 20, 26, 27, 32, 34, 13, 12, 16, 17, 16, 35. Per 2 times (0.2%)

Os gráficos gerados são irrelevantes, apenas indicam que todas as simulações terminaram no local final, por esse motivo foram colocados juntos na Figura 96. O cenário construído não possui dados de atributos ou batalhas, portanto não existem os gráficos relativos a esses aspectos. Os demais se encontram a seguir.

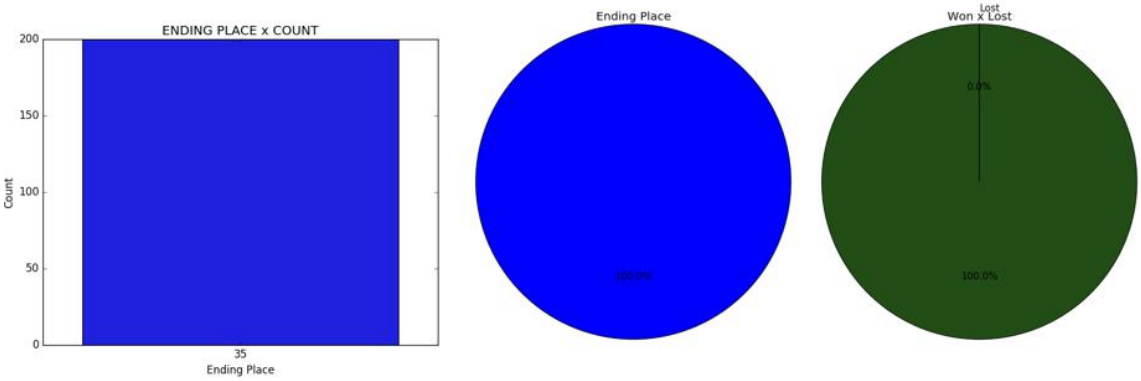

Figura 96: Gráficos das simulações para *Resident Evil 2*

Super Mario Bros 3

O arquivo XML a seguir corresponde ao mapa da Figura 61, representando o primeiro mundo de *Super Mario Bros 3*.

```
<root>
 <world>
  <place id='1' neighbors='2'/>
  <place id='2' neighbors='3,9'/>
  <place id='3' neighbors='2,4,5,7'/>
  <place id='4' neighbors='3,5,6'/>
  <place id='5' neighbors='3,4,6,7'/>
  <place id='6' neighbors='4,5'>
```

```
    <item id='Toad house item' type='atk' value='10' quantity='1'/>
   </place>
   <place id='7' neighbors='3,5,8'>
    <item id='Luck game item' type='lif' value='10' quantity='1' />
   </place>
   <place id='8' neighbors='7,9,10'>
    <item id='Fortress Key' type='key'/>
   </place>
   <place id='9' neighbors='2,8' idLock='Fortress Key'/>
   <place id='10' neighbors='8,11'/>
   <place id='11' neighbors='10,12,13'/>
   <place id='12' neighbors='11'>
    <item id='Toad house item' type='dfs' value='10' quantity='1'/>
   </place>
   <place id='13' neighbors='11' final='true'/>
  </world>
</root>
```

Os relatórios resumido e detalhado possuem respectivamente 50.159 e 81.930 linhas, para mil simulações. Sua síntese se encontra a seguir.

```
----------Simulation report after 1000 expeditions:----------
Heroes successes: 1000 (100.0%)
Heroes failures: 0 (0.0%)
Avg visited places: 17.477
Avg items found: 3.183
Turns avg per expedition: 0.0
Turns avg per places: 0.0
Most common visit order: 1, 2, 3, 4, 6, 5, 7, 8, 10, 11, 13. Per 70 times (7.0%)
```

Assim como no exemplo anterior para o jogo *Resident Evil 2*, a única informação a ser extraída dos gráficos é que todas as simulações foram finalizadas no local final, nesse caso identificado com o número 13. Por esse motivo os gráficos não foram incluídos.

Diablo

O arquivo XML a seguir corresponde ao primeiro conjunto de mil simulações executadas em *Diablo*. O mapa do mundo de jogo é equivalente ao da Figura 63.

```
<root>
 <world>
  <place id='1' neighbors='2'>
   <foe name='Monster' lif='20' atk='6' dfs='2' agi='6' quantity='2'/>
  </place>
  <place id='2' neighbors='3'>
   <foe name='Monster' lif='20' atk='6' dfs='2' agi='6' quantity='4'/>
```

```
  <item id='potion' type='lif' value='30' quantity='1' />
</place>
<place id='3' neighbors='4'>
  <foe name='Monster' lif='20' atk='6' dfs='2' agi='6' quantity='6'/>
</place>
<place id='4' neighbors='5'>
  <foe name='Monster' lif='20' atk='6' dfs='2' agi='6' quantity='8'/>
  <item id='potion' type='lif' value='30' quantity='1' />
</place>
<place id='5' neighbors='6'>
  <foe name='Monster' lif='20' atk='6' dfs='2' agi='6' quantity='10'/>
</place>
<place id='6' neighbors='7'>
  <foe name='Monster' lif='20' atk='6' dfs='2' agi='6' quantity='12'/>
  <item id='potion' type='lif' value='30' quantity='1' />
</place>
<place id='7' neighbors='8'>
  <foe name='Monster' lif='20' atk='6' dfs='2' agi='6' quantity='14'/>
</place>
<place id='8' neighbors='9'>
  <foe name='Monster' lif='20' atk='6' dfs='2' agi='6' quantity='16'/>
  <item id='potion' type='lif' value='40' quantity='1' />
</place>
<place id='9' neighbors='10'>
  <foe name='Monster' lif='20' atk='6' dfs='2' agi='6' quantity='18'/>
</place>
<place id='10' neighbors='11'>
  <foe name='Monster' lif='20' atk='6' dfs='2' agi='6' quantity='20'/>
  <item id='potion' type='lif' value='40' quantity='1' />
</place>
<place id='11' neighbors='12'>
  <foe name='Monster' lif='20' atk='6' dfs='2' agi='6' quantity='22'/>
</place>
<place id='12' neighbors='13'>
  <foe name='Monster' lif='20' atk='6' dfs='2' agi='6' quantity='24'/>
  <item id='potion' type='lif' value='40' quantity='1' />
</place>
<place id='13' neighbors='14'>
  <foe name='Monster' lif='20' atk='6' dfs='2' agi='6' quantity='26'/>
</place>
<place id='14' neighbors='15'>
  <foe name='Monster' lif='20' atk='6' dfs='2' agi='6' quantity='28'/>
  <item id='potion' type='lif' value='50' quantity='1' />
</place>
<place id='15' neighbors='16'>
  <foe name='Monster' lif='20' atk='6' dfs='2' agi='6' quantity='30'/>
  <item id='potion' type='lif' value='50' quantity='1' />
</place>
<place id='16' final='true'>
  <foe name='Diablo' lif='300' atk='16' dfs='8' agi='4'/>
</place>
```

</world>
</root>

Os relatórios resumido e detalhado, gerados após mil simulações, possuem respectivamente 50.657 e 2.690.079 linhas. Sua síntese está abaixo.

----------Simulation report after 1000 expeditions:----------
Heroes successes: 0 (0.0%)
Heroes failures: 1000 (100.0%)
Avg visited places: 11.575
Avg items found: 4.922
Turns avg per expedition: 2453.09
Turns avg per places: 211.68
Most common visit order: 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12. Per 653 times (65.3%)

Os gráficos gerados são reproduzidos a seguir.

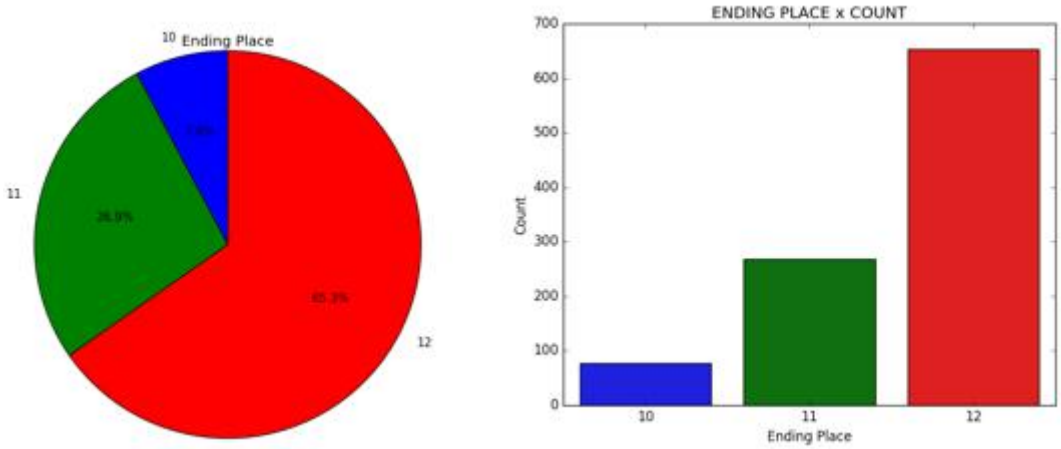

Figura 97: Local fim da simulação

Figura 98: Dispersão do atributo vida do herói

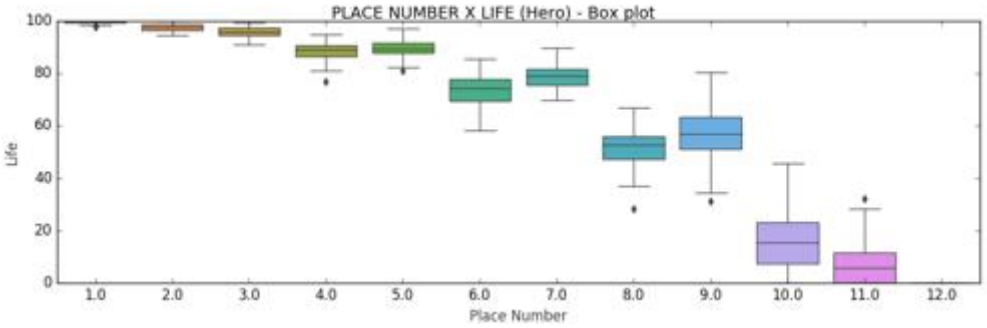

Figura 99: *Box plot* de variação do atributo vida do herói

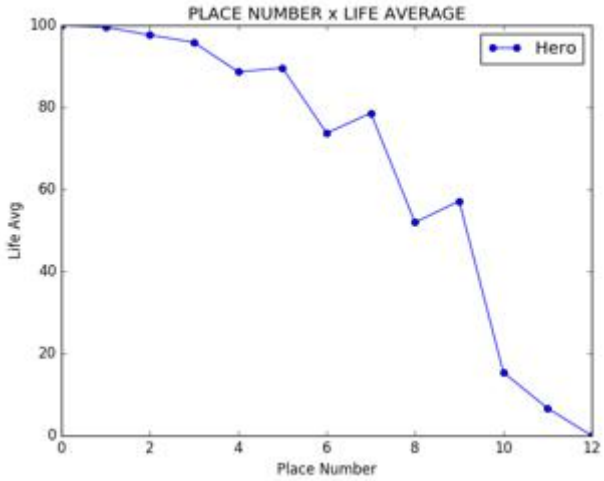

Figura 100: Média do atributo vida do herói

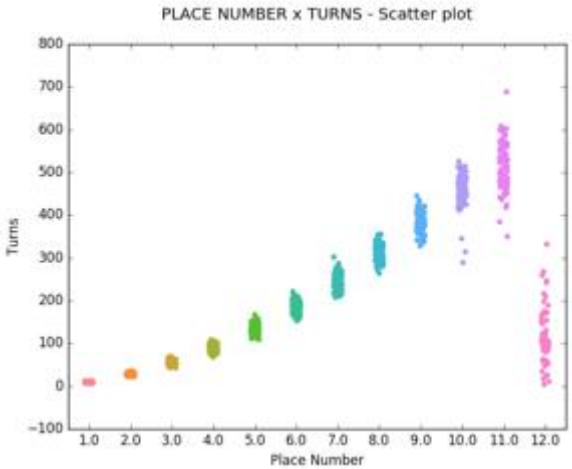

Figura 101: Dispersão da quantidade de turnos por local

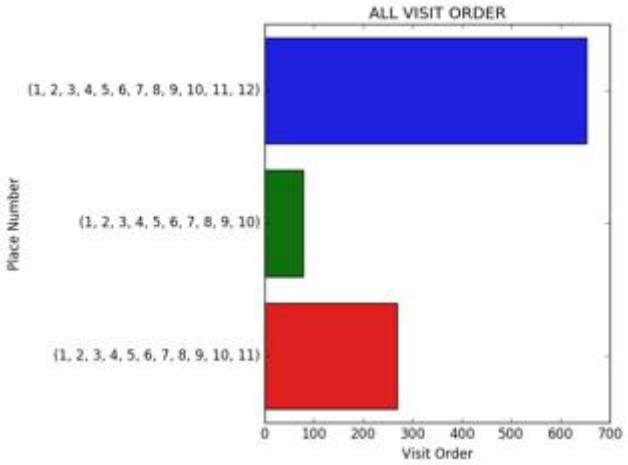

Figura 102: Ordem de visita aos locais

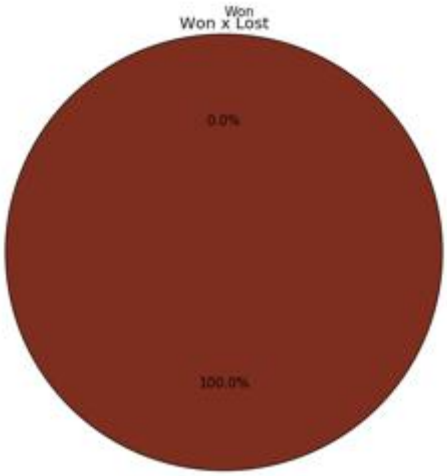

Figura 103: Sucesso e falha

Após modificações, houve um segundo conjunto de simulações. O arquivo XML correspondente é mostrado a seguir.

```
<root>
 <world>
  <place id='1' neighbors='2'>
   <foe name='Monster' lif='20' atk='6' dfs='2' agi='6' quantity='1'/>
  </place>
  <place id='2' neighbors='3'>
   <foe name='Monster' lif='20' atk='6' dfs='2' agi='6' quantity='3'/>
   <item id='potion' type='lif' value='30' quantity='1' />
  </place>
  <place id='3' neighbors='4'>
   <foe name='Monster' lif='20' atk='6' dfs='2' agi='6' quantity='5'/>
   <item id='sword' type='atk' value='2' quantity='1' />
```

```
</place>
<place id='4' neighbors='5'>
  <foe name='Monster' lif='20' atk='6' dfs='2' agi='6' quantity='7'/>
  <item id='potion' type='lif' value='30' quantity='1' />
</place>
<place id='5' neighbors='6'>
  <foe name='Monster' lif='20' atk='6' dfs='2' agi='6' quantity='9'/>
  <item id='sword' type='atk' value='2' quantity='1' />
</place>
<place id='6' neighbors='7'>
  <foe name='Monster' lif='20' atk='6' dfs='2' agi='6' quantity='11'/>
  <item id='potion' type='lif' value='30' quantity='1' />
</place>
<place id='7' neighbors='8'>
  <foe name='Monster' lif='20' atk='6' dfs='2' agi='6' quantity='13'/>
  <item id='sword' type='atk' value='2' quantity='1' />
</place>
<place id='8' neighbors='9'>
  <foe name='Monster' lif='20' atk='6' dfs='2' agi='6' quantity='15'/>
  <item id='potion' type='lif' value='40' quantity='1' />
</place>
<place id='9' neighbors='10'>
  <foe name='Monster' lif='20' atk='6' dfs='2' agi='6' quantity='16'/>
  <item id='sword' type='atk' value='2' quantity='1' />
</place>
<place id='10' neighbors='11'>
  <foe name='Monster' lif='20' atk='6' dfs='2' agi='6' quantity='19'/>
  <item id='potion' type='lif' value='40' quantity='1' />
</place>
<place id='11' neighbors='12'>
  <foe name='Monster' lif='20' atk='6' dfs='2' agi='6' quantity='21'/>
  <item id='sword' type='atk' value='2' quantity='1' />
</place>
<place id='12' neighbors='13'>
  <foe name='Monster' lif='20' atk='6' dfs='2' agi='6' quantity='23'/>
  <item id='potion' type='lif' value='40' quantity='1' />
</place>
<place id='13' neighbors='14'>
  <foe name='Monster' lif='20' atk='6' dfs='2' agi='6' quantity='25'/>
  <item id='sword' type='atk' value='2' quantity='1' />
</place>
<place id='14' neighbors='15'>
  <foe name='Monster' lif='20' atk='6' dfs='2' agi='6' quantity='27'/>
  <item id='potion' type='lif' value='50' quantity='1' />
</place>
<place id='15' neighbors='16'>
  <foe name='Monster' lif='20' atk='6' dfs='2' agi='6' quantity='29'/>
  <item id='sword' type='dfs' value='2' quantity='1' />
</place>
<place id='16' final='true'>
  <foe name='Diablo' lif='300' atk='16' dfs='8' agi='4'/>
```

```
    </place>
  </world>
</root>
```

Os relatórios resumido e detalhado, gerados após mil simulações, possuem respectivamente 73.010 e 2.970.373 linhas. Sua síntese está abaixo.

----------Simulation report after 1000 expeditions:----------
Heroes successes: 7 (0.7%)
Heroes failures: 993 (99.3%)
Avg visited places: 16.0
Avg items found: 14.0
Turns avg per expedition: 2623.36
Turns avg per places: 163.96
Most common visit order: 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16. Per 1000 times (100.0%)

Os gráficos gerados são reproduzidos a seguir.

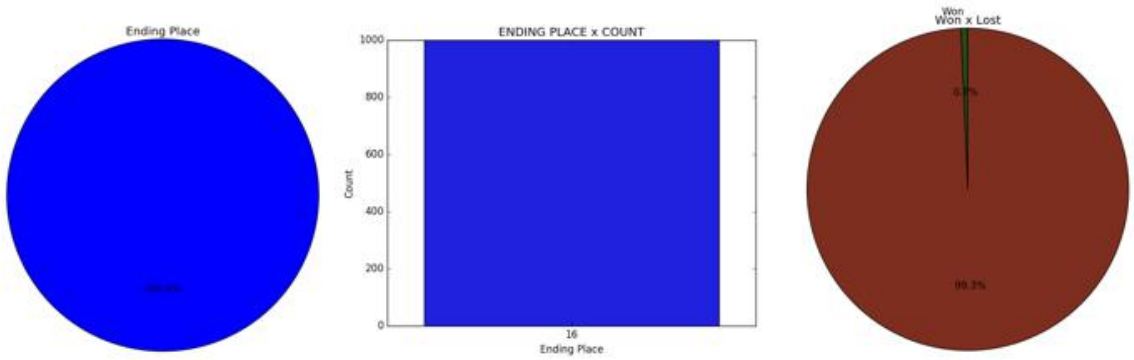

Figura 104: Local final e Sucesso e falha

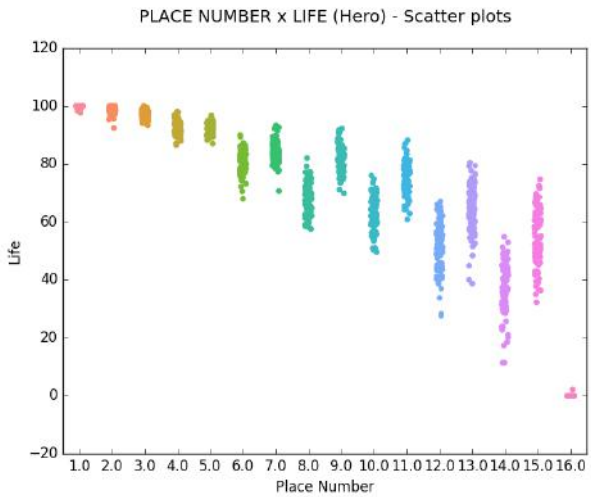

Figura 105: Dispersão do atributo vida do herói

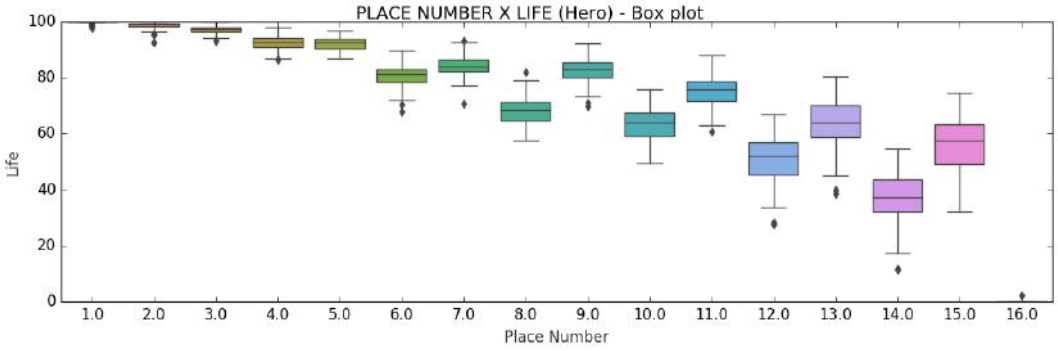

Figura 106: *Box plot* de variação do atributo vida do herói

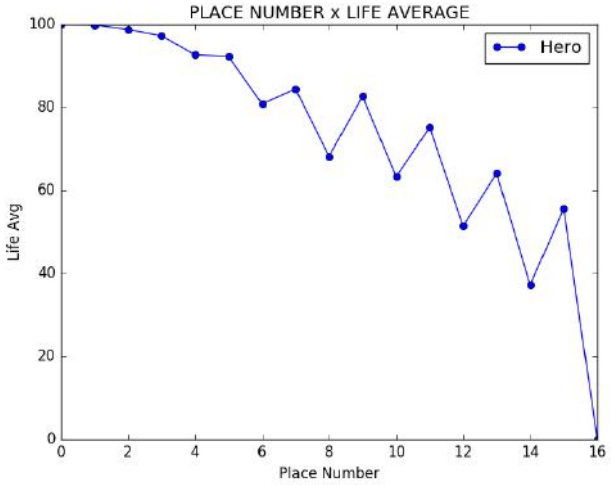

Figura 107: Média do atributo vida do herói

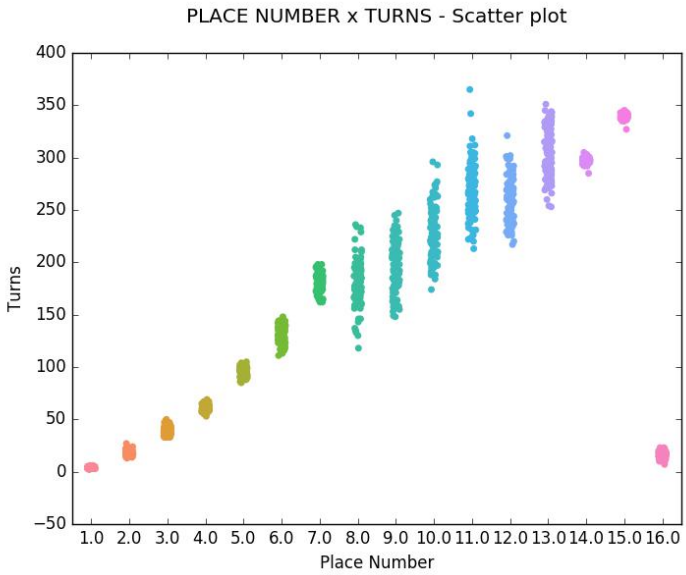

Figura 108: Dispersão da quantidade de turnos por local