AVALIAÇÃO AUTOMATIZADA DE ESTÉTICA DE JOGOS GENÉRICOS DE CARTAS E DADOS

Augusto Acioli Pinho Vanderley

Dissertação de Mestrado 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 Mestre em Engenharia de Sistemas e Computação.

Orientador: Geraldo Bonorino Xexéo

Rio de Janeiro
Setembro de 2018

AVALIAÇÃO AUTOMATIZADA DE ESTÉTICA DE JOGOS GENÉRICOS DE CARTAS E DADOS

Augusto Acioli Pinho Vanderley

DISSERTAÇÃO 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 MESTRE EM CIÊNCIAS EM ENGENHARIA DE SISTEMAS E COMPUTAÇÃO.

Examinada por:

Prof. Geraldo Bonorino Xexéo, D.Sc.

Prof. Jano Moreira de Souza, Ph.D.

Prof. Adriana Santarosa Vivacqua, D.Sc.

RIO DE JANEIRO, RJ – BRASIL
SETEMBRO DE 2018

Vanderley, Augusto Acioli Pinho

Avaliação Automatizada de Estética de Jogos Genéricos de Cartas e Dados/Augusto Acioli Pinho Vanderley. – Rio de Janeiro: UFRJ/COPPE, 2018.

XIII, 72 p.: il.; 29, 7cm.

Orientador: Geraldo Bonorino Xexéo

Dissertação (mestrado) – UFRJ/COPPE/Programa de Engenharia de Sistemas e Computação, 2018.

Referências Bibliográficas: p. 64 – 66.

1. avaliação de jogos. 2. linguagem de descrição de jogos. 3. avaliação de qualidade. 4. jogos de dados. I. Xexéo, Geraldo Bonorino. II. Universidade Federal do Rio de Janeiro, COPPE, Programa de Engenharia de Sistemas e Computação. III. Título.

Dedicado aos meus pais
pela confiança.

Agradecimentos

Agradeço principalmente ao meu orientador, que me acolheu em momentos de desespero, confiou em mim até o momento final e sem o qual esse trabalho não seria possível. Sua visão objetiva e pontual foram essenciais para o andamento do projeto. Sua dedicação e expertise o destacam como um exemplo de profissional a ser seguido. Desta forma, muito obrigado Geraldo Xexéo.

Agradeço aos meus amigos e companheiros do LUDES, de onde diversas ideias e críticas surgiram. Dentre diversos participantes importantes, cito principalmente Eduardo Mangeli, cuja dissertação serviu como base para o trabalho, sendo a pessoa que eu mais incomodei com dúvidas.

Agradeço aos amigos do laboratório CAPGOV que vivenciaram comigo essa longa jornada da pós-graduação. Dentre eles, um agradecimento especial para Xiao Yuan Kong, Débora Lima, Juan Baptista, Bernardo Blasquez, Douglas Paranhos e Romulo Freires.

Agradeço aos amigos da UFRJ que perduram em minha vida e me deram forças para continuar. Alguns destes são Gabriel Moraes, Lucian Sturião, Caio Moraes. Agradeço também

Agradeço aos amigos e colegas que não foram citados aqui mas que colaboraram de alguma forma com esse trabalho, direta ou indiretamente.

Agradeço por fim aos meus pais, pela imensa confiança que depositaram em mim, por sempre acreditarem e confiarem na minha capacidade e por serem as melhores pessoas que eu conheço, tanto individualmente quando como casal.

O presente trabalho foi realizado com apoio do CNPq, Conselho Nacional de Desenvolvimento Científico e Tecnológico - Brasil.

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

AVALIAÇÃO AUTOMATIZADA DE ESTÉTICA DE JOGOS GENÉRICOS DE CARTAS E DADOS

Augusto Acioli Pinho Vanderley

Setembro/2018

Orientador: Geraldo Bonorino Xexéo

Programa: Engenharia de Sistemas e Computação

Entender o sentimento que o ato de jogar proporciona é uma questão fundamental para o design de jogos. Pelo processo de playtest ser difícil e custoso, o Design Automatizado de Jogos tem sido uma área importante para o desenvolvimento de novos jogos. O trabalho aqui presente oferece uma ferramenta para a avaliação automatizada de jogos de cartas e dados. Assim, este trabalho apresenta RECYCLEDICE, uma linguagem que permite descrever jogos de cartas e dados. Também foi adaptada uma ferramenta para interpretar e simular jogos, jogando automaticamente e gerando estatísticas de jogo. Por fim, combinou-se as simulações com métricas de estética, como drama, mudança de liderança e incerteza, para assim gerar informações relevantes sobre o jogo.

Abstract of Dissertation presented to COPPE/UFRJ as a partial fulfillment of the requirements for the degree of Master of Science (M.Sc.)

AUTOMATIC EVALUATION OF AESTHETICS IN GENERIC CARD AND DICE GAMES

Augusto Acioli Pinho Vanderley

September/2018

Advisor: Geraldo Bonorino Xexéo

Department: Systems Engineering and Computer Science

Understanding the feeling that playing a game provides to the player is quintessential to game design. As playtesting is difficult and expensive, Automatic Game Design has been an area that keeps growing and getting much attention. The present work aims to offer a tool for automatic avaliation of dice and card games. Therefore, this work presents RECYCLEDICE, a language that allows card and dice games to be descripted. Furthermore, a tool was adapted to interpret and simulate games, playing automatically and generating statistics from gameplaying. Lastly, simulations and aesthetics metrics, such as drama, lead change and uncertainty, were combined in order to generate relevant information about the game.

Sumário

Lista de Figuras

Lista de Tabelas

Capítulo 1

Introdução

Esse capítulo expõe a motivação para a pesquisa, mostrando o problema da pesquisa e descrevendo de forma breve o trabalho. A última seção apresenta a estrutura do documento.

1.1 Motivação

A indústria de jogos é uma das que mais cresce no mundo. Em 2012, o faturamento anual da indústria de jogos digitais foi de 70.6 bilhões de dólares. Já em 2021, este valor está previsto para 180.1 bilhões de dólares (WIJMAN, 2018). Jogos têm se mostrado presentes no dia a dia de milhões de pessoas, seja como forma de facilitar aprendizado, como gamificação de processos ou como um simples passatempo. Assim sendo, o design de jogos se revela uma área promissora, ao mesmo tempo que representa uma área desafiadora.

O design de jogos é uma área complexa. As regras de um jogo precisam ser desenvolvidas de forma que produzam interesse e diversão no jogador. Essa relação entre mecânicas e sentimento do jogador ao jogar não é algo trivial de ser avaliada. Busca-se uma melhor compreensão das interações humanas com o ato de jogar, procurando entender o que o jogador sente e o que torna um jogo interessante para ele. O *framework MDA* (HUNICKE *et al.*, 2004) foi proposto com o intuito de formalizar e especificar essa relação. Na perspectiva do designer, as mecânicas do jogo geram dinâmicas de sistema, que levam a experiências de estética específicas. Assim, qualquer mudança na mecânica acarreta em uma dinâmica diferente, proporcionando uma alteração na estética, como indicado na figura 1.1. A partir deste modelo, percebe-se a importância de pensar em cada um dos componentes do jogo durante sua criação.

Tipicamente, jogos são desenvolvidos por um designer humano, que modela e cria as regras através de sua experiência e intuição. A partir do momento que existe um jogo testável, jogadores de teste avaliam o jogo e respondem sobre sua experiência,

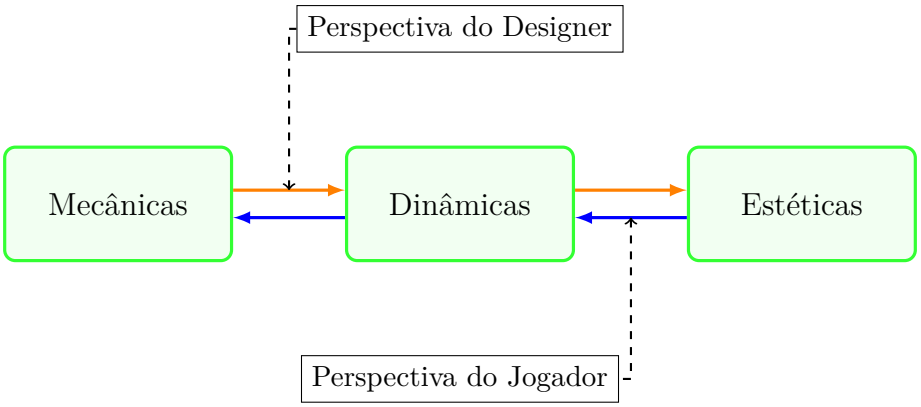

Figura 1.1 – Diagrama MDA

permitindo assim que o designer desenvolva melhor o jogo com base nessa respostas, em um processo iterativo (FULLERTON, 2008). A figura 1.2 mostra um ciclo de desenvolvimento iterativo. Primeiramente, o designer cria e desenvolve as regras do seu jogo, além de gerar um protótipo inicial para ser testado. A partir deste protótipo, um *playtest* é realizado, que consiste em colocar pessoas para jogar com o objetivo de descobrir se o jogador está tendo a experiência desejada. Métricas sobre o *playtest* são salvas e analisadas para entender melhor a experiência do jogador. Com base nessas métricas, o designer reescreve as regras e reajusta parâmetros, gerando um novo protótipo. Esse ciclo se repete, com *playtests* sendo realizado novamente, até que o jogo esteja na forma desejada.

Figura 1.2 – Modelo de Desenvolvimento Padrão de Jogos

Contudo, esse ciclo de desenvolvimento iterativo com *playtest* apresenta algumas desvantagens. De imediato, pode-se citar o tempo excessivo demandado para realizar esse ciclo: *playtests* são demorados e a análise manual das estatísticas para reescrever as regras demanda um tempo elevado. Depois, têm-se o custo elevado para realização dos *playtests*, visto que cada iteração demanda um *playtest* com jogadores pagos. Também existe a substituição dos *playtesters* ao longo dos ciclos, seja pelo cansaço, pela impossibilidade de continuar testando ou pela familiarização com o jogo, acarretando na busca por novos jogadores. Por fim, a amostragem de

jogadores deve ser bem escolhida, visto que uma triagem mal feita pode gerar uma representatividade pequena dos perfis de jogadores.

Para solucionar essas dificuldades, uma nova área foi cunhada, o design automatizado de jogos, que consiste na criação de novos jogos a partir de um processo iterativo em que todos os componentes do processo são automatizados: *playtest*, avaliação das estatísticas e regulagem das regras. A imagem 1.3 mostra o processo. Assim sendo, a função do designer passa a ser de supervisionar o processo pela visão humana, evitando erros comuns à máquina. Nesta área, o primeiro passo é a criação e descrição de uma linguagem específica para os jogos que se deseja modelar. Em seguida, descreve-se algum jogo existente usando a linguagem previamente definida, modelando as regras. Ao mesmo tempo, um sistema precisa ser criado para exercer duas funcionalidades importantes: interpretar e simular o jogo a partir de sua descrição e realizar *playtests* com jogadores automatizados. Posteriormente, após estatísticas dos jogos serem geradas, esses dados são passados para um avaliador cujo trabalho é, através de métricas de avaliação, contabilizar o quão interessante foi o jogo em determinados aspectos. Por fim, novos programas de descrição podem ser gerados automaticamente, usando a mutação do programa original a partir das métricas de avaliação como guia. Essas variantes implicam na repetição de todo o processo de simulação, *playtest* e avaliação, formando um ciclo de desenvolvimento de jogos.

É importante conceituar que os humanos continuam sendo imprescindíveis à produção de um jogo. Ao final do processo e do ciclo automatizado, tendo em mãos um jogo que se adapta melhor às métricas, *playtests* com pessoas devem ser realizados, com o objetivo de obter uma avaliação e um *tuning* final mais corretos. Desta forma, esse ciclo economiza *playtests* com pessoas ao encontrar automaticamente erros de design, mas a interação do jogo com pessoas continua sendo um componente essencial para o design de um jogo. Segundo ALTHÖFER (2003), computadores correspondem a uma ferramenta para auxiliar na criação de jogos e sempre será necessário uma mente humana criativa para realizar esta tarefa.

A tese de doutorado de (BROWNE, 2008) foi pioneira na área, visto que seu autor criou uma linguagem de descrição de jogos, implementou um sistema que joga automaticamente, criou as diferentes métricas para avaliação, além de gerar novos jogos através da evolução de regras. Contudo, seu sistema não está disponível, não sendo possível realizar trabalhos adicionais sobre o mesmo[1].

Perseguiu-se assim um projeto de sistema onde o usuário seria capaz de escrever seus próprios jogos, realizar as simulações e por fim avaliar o jogar. Esse trabalho não aborda a questão da geração automática de novos jogos, mantendo como foco

[1]Contactado, o autor não permitiu o uso de seu sistema

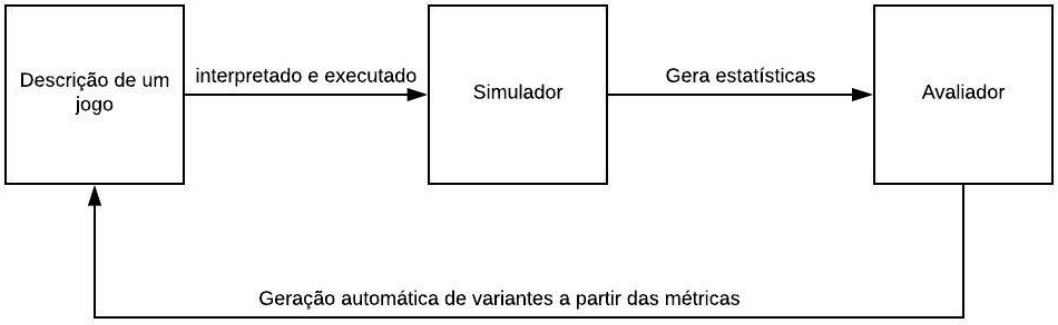

Figura 1.3 – Modelo de Desenvolvimento Automatizado de Jogos

os outros três componentes essenciais para que a descoberta de variantes possa ser realizada.

1.2 Problema da Pesquisa

A proposta desta dissertação foi de conectar um simulador de jogos de cartas e dados a um avaliador de jogos em turno, realizando modificações e melhorias necessárias para poder assim responder ao problema da pesquisa:

- É possível descrever jogos de cartas e dados de forma que possam ser avaliados automaticamente por um *framework* de avaliação de estética?

1.3 Metolodologia

O problema da pesquisa implicou nas seguintes perguntas de pesquisa, que precisam ser respondidas.

- Qual linguagem deve ser utilizada?
- Como a linguagem será interpretada e simulada?
- Qual *framework* de avaliação será escolhido?
- Como extrair as estatísticas das simulações e fornecer esses dados para a ferramenta de avaliação?
- Como garantir que as métricas estão condizentes com as simulações?

Para este trabalho, foi encontrada uma linguagem que supre os requisitos de balanço entre generalidade e especificidade. Assim optou-se pelo estudo da linguagem de jogos de cartas, um gênero popular que permite modelar um conjunto numeroso

de jogos. Uma linguagem de descrição de jogos de cartas, RECYCLE, proposta por (BELL & GOADRICH, 2016), serviu como base para o desenvolvimento da nova linguagem.

Além disso, escolheu-se pelo *simulador* CARDSTOCK (BELL & GOADRICH, 2016), capaz de interpretar a linguagem RECYCLE e de realizar simulações com jogadores aleatórios e inteligentes. O elemento primordial para a escolha do *framework* foi que ele possui código aberto, sendo assim passível de mudanças.

Em relação ao avaliador, o *framework* proposto por (MANGELI, 2016) mostrou-se flexível para inserção de novos jogos para serem avaliados, assim como a possibilidade de implementação de novas métricas de estética. Ademais, MANGELI (2016) realizou sua pesquisa com jogos de campeonatos reais, com dados de jogadores pessoas, e indicou como trabalho futuro a aplicação do seu *framework* a jogos provenientes de simulações.

Para extrair informação das partidas, uma estrutura de avaliação foi proposta. Essa arquitetura salva estatísticas das partidas e as apresenta para a ferramenta de avaliação, no formato de pontuação por turno.

Por fim, uma avaliação da distribuição das métricas em um conjunto de partidas simuladas permitiu verificar o funcionamento da arquitetura.

1.4 Escopo

- Desejou-se estender a linguagem RECYCLE, permitindo que dados jogáveis fossem adicionados como funcionalidade em jogos de cartas.
- Também foi necessário que o CARDSTOCK fosse responsável pela exportação dos dados de pontuação a cada turno, utilizados no avaliador.
- Outra contribuição importante foi de adaptar o avaliador para que este pudesse aceitar os novos jogos de forma genérica, modelando um único jogo genérico para o avaliador.
- A partir disso, jogos puderam ser avaliados e resultados foram gerados quanto às métricas do jogo.
- Por fim, desejou-se avaliar o comportamento do avaliador e de suas métricas ao serem introduzidos a um grande número de simulações de jogos.

1.5 Estrutura do Documento

Este trabalho compreende quatro capítulos além desse.

O capítulo 2 apresenta fundamentos teóricos necessários para o entendimento e desenvolvimento da dissertação. Para isso, um *framework* que relaciona mecânica e estéticas é apresentado, uma visão ampla sobre General Game Playing é abordada, linguagens de descrição de jogos são mostradas, com um foco no RECYCLE. Posteriormente, a interpretação de uma gramática é abordada, apresentando a ferramenta CARDSTOCK. Mecânicas de jogos de tabuleiros importantes para fundamentar o trabalho são apresentadas. Uma conceituação sobre as métricas de estética é abordada e, por fim, trabalhos relacionados são discutidos.

O capítulo 3 traz as contribuições desenvolvidas para resolver o problema da pesquisa. Desta forma, apresenta o modelo de estrutura da avaliação automatizada da estética.

O capítulo 4 detalha a implementação das ideias desenvolvidas neste trabalho. Ele mostra o desenvolvimento do RECYCLEDICE, as modificações no simulador e a adaptação da *framework de avaliação* para que a estrutura fosse possível. Também descreve a implementação de dois jogos descritos na nova linguagem.

O capítulo 5 compreende a avaliação da solução proposta e implementada no capítulo anterior. Desta forma, apresenta os resultados como prova de conceito, além de verificar o funcionamento das métricas a partir de variantes de dois jogos e por fim verificar a melhor variante para cada um deles.

Por fim, o capítulo 6 conclui o trabalho com uma revisão dos objetivos e como eles foram concluídos, assim como descreve trabalhos futuros.

Capítulo 2

Fundamentação Teórica

Este capítulo apresenta a fundamentação teórica necessária para entendimento dos conceitos desenvolvidos nesta dissertação. Assim, ele introduz o modelo MDA, estabelece uma base de conhecimento em relação a linguagem de descrições de jogos, descreve a linguagem RECYCLE usada na implementação, apresenta uma fundamentação sobre árvore de análise sintática para assim explicar o simulador CARDSTOCK. O capítulo também fundamenta as métricas de estéticas aplicadas ao problema, apresentando as métricas presentes no *framework* de avaliação de MANGELI. Por fim, o capítulo expõe os trabalhos relacionados na área.

2.1 Framework MDA

Cunhada por HUNICKE *et al.* (2004), MDA, sigla para Mecânicas, Dinâmicas e Estéticas, é uma metodologia para o design de jogos. Possui como objetivo estabelecer conceitos para ajudar desenvolvedores e designers de jogos a se entenderem melhor, através de uma metodologia única, decompondo o jogo.

O modelo MDA propõe três componentes distintos:

- Mecânicas: os componentes fundamentais do jogo, todos os elementos que fazem parte de seu universo, a nível de representação de dados e algoritmos. Regras, estados do jogo, movimentos possíveis e narrativa consistem em alguns exemplos de mecânica.

- Dinâmicas: correspondem ao comportamento em tempo real das mecânicas, o efeito que elas geram no jogo.

- Estética: A resposta emocional sentida pelo jogador ao interagir com o sistema.

Ao criar um jogo, o designer desenvolve as mecânicas, definindo ações possíveis e comportamento desejado. O designer de jogos possui acesso somente às mecânicas, dado que este é o único elemento que se pode alterar diretamente no desenvolvimento

de um jogo. As dinâmicas e as estéticas não podem ser construídas, são elementos conceituais que o designer não possui acesso. Considere um jogo de ação de tiro em primeira pessoa, uma mecânica possível de estabelecer é correr. Correr é uma mecânica que possui diferentes parâmetros, como velocidade e aceleração. Todos os elementos do correr são programados diretamente.

Assim, através das mecânicas um designer de jogos consegue conceber as dinâmicas, que consistem no modo como os elementos da mecânica se relacionam com as interações do jogador, por exemplo as estratégias que surgem destas regras. Dado a mecânica de corrida, uma dinâmica resultante é de fugir. Não se programa o fugir diretamente, mas a interação do jogador com um monstro veloz e a opção de correr permite que a dinâmica seja desenvolvida indiretamente.

Por fim, o próprio ato de fugir implica em uma experiência sentimental para o jogador, que pode ser traduzido em medo. Essa interação entre o jogador e as dinâmicas que gera essa resposta emocional no jogador, a estética. Segundo HUNICKE *et al.* (2004), a taxonomia não restrita da estética consiste em oito sentimentos, apresentados na tabela 2.1.

1. **Sensação** Jogo como prazer sensorial.	2. **Fantasia** Jogo como faz-de-conta.
3. **Narrativa** Jogo como drama.	4. **Desafio** Jogo como obstáculo.
5. **Companheirismo** Jogo como ambiente social.	6. **Descoberta** Jogo como território desconhecido
7. **Expressão** Jogo como auto-expressão.	8. **Submissão** Jogo como passatempo

Tabela 2.1 – Taxonomia da Estética do MDA

Assim, busca-se, através da estética, sentimentos que sejam interessantes para o público alvo.

Para o designer de jogos, a interação do jogador com as mecânicas geram as dinâmicas, que por sua vez fornece as experiências de estética do jogador. Em contrapartida, do ponto de vista do jogador, a estética é a impressão que ele tem, a partir das dinâmicas observáveis e da interação com as mecânicas. A figura 1.1 representa as diferentes perspectivas que jogador e designer possuem. É interessante pensar nas duas perspectivas, visto que alterações em um dos componentes acarretam automaticamente em mudanças nos outros.

Considerando a importância de um bom conjunto de mecânicas, isso se traduz na importância de construir regras bem definidas e de modelar elas. Essa modelagem pode ser realizada através de uma linguagem bem estruturada, uma linguagem de descrição de jogos.

2.2 General Game Playing

Um marco na história do desenvolvimento de um jogador automático de xadrez ocorreu em 1999, quando um computador mostrou-se capaz de jogar xadrez eficientemente. Para tanto, a inteligência artificial deveria ser capaz de vencer um campeonato contra o campeão mundial de xadrez em um tempo limite predeterminado, permitindo somente três minutos por jogada. Com o intuito de vencer o campeonato, Deep Blue (HSU, 1999) foi desenvolvido pela IBM, mostrando-se capaz de vencer três jogos dentre seis, empatando um e perdendo outros dois, obtendo assim uma pontuação final de 3.5 a 2.5. Como consequência, superou-se o desafio de criar um agente inteligente para jogos de Xadrez.

Além do Xadrez, diversos estudos foram realizados para a criação de agentes inteligentes em jogos. Cada um destes agentes foi idealizado e construído com o intuito de resolver um problema específico em relação à um jogo, ou seja, jogar cada jogo de forma eficiente. Dentre os jogos mais conhecidos, alguns exemplos de agentes inteligentes especialistas podem ser citados:

- Jogo da velha (FOGEL, 1993), Damas (SCHAEFFER *et al.*, 2007) e Othello (VAN ECK & VAN WEZEL, 2008), representam classes de jogos multijogadores e com informação completa, ou seja, onde toda informação dos estados do jogo é conhecida a cada turno pelos jogadores.

- Gamão (TESAURO, 2002), representando uma classe de jogos de informação imperfeita, na qual existe aleatoriedade.

- Poker (BOWLING *et al.*, 2015) e MahJong (DE BONDT, 2012), jogos com informação incompleta, onde existe informação escondida para cada jogador.

Contudo, esta abordagem de agentes especializados ignora um aspecto importante no estudo da Inteligência Artificial, que consiste na generalização: o Deep Blue não entende as regras do Jogo da Velha ou Damas, assim como um agente criado para o jogo da Velha não é capaz de realizar as ações possíveis no Xadrez. O conceito de uma máquina inteligente não especializada e capaz de tomar boas decisões permanecia no imaginário científico. A imagem 2.1 mostra um exemplo de como, para jogar cada jogo, é necessário ter um agente específico.

Tendo esse desafio em mente, ocorreu o surgimento da área de General Game Playing (GGP), um projeto iniciado pelo grupo de lógica de Stanford em 2005 (GENESERETH *et al.*, 2005), cujo objetivo consiste em desenvolver agentes aptos a jogar uma classe de jogos, de forma eficiente, ao invés de se especializar em um único jogo. Este agente não conhece de antemão os jogos que lhe são apresentados, não podendo assim depender de algoritmos específicos para jogar. A figura 2.2

Figura 2.1 – Exemplo de Agentes Especialistas

exemplifica o comportamento de um Agente de General Game Playing. As regras do jogo são passadas para um gerenciador de Jogos Genéricos, que interpreta a regra e transmite ao agente as possíveis ações, que responde com sua decisão. Em seguida, as informações da partida são registradas, com o intuito de verificar se o agente jogou eficientemente os diferentes jogos.

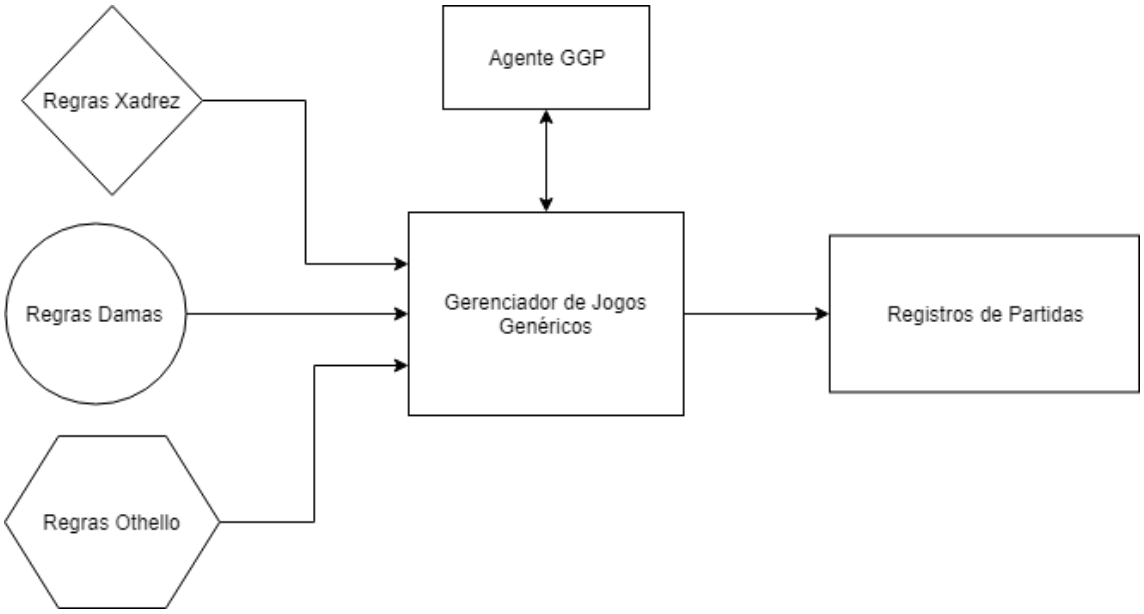

Figura 2.2 – Exemplo de Agente de General Game Playing

General Game Players são sistemas que não conhecem as regras de um jogo previamente e conseguem jogá-lo em tempo real estrategicamente sem intervenção humana. Descrições de um jogo são transmitidas para este sistema, que é responsável por traduzir as regras, determinar a melhor jogada e responder com uma ação. Os jogos são modelados a partir de uma linguagem chamada Game Description Language, que permite representar as regras do jogo.

2.3 Game Description Language

GDL (Game Description Language) é uma linguagem formal para definir jogos. A GGP utiliza uma linguagem própria, nomeada de GDL, que, para não gerar uma ambiguidade com o conceito de GDL, chamamos de Stanford GDL (LOVE *et al.*, 2008). A GDL de Stanford descreve o estado do jogo como uma série de fatos e as mecânicas, ou seja as funções de transição entre estados, como regras lógicas. Ela permite que uma classe de jogos seja descrita de forma que agentes jogadores de GGP possam jogar qualquer jogo descrito nesta linguagem.

Essa linguagem descreve uma classe de jogos: finitos, discretos, determinísticos, multi jogadores e de informação completa.

- Finito: Um final é alcançado após um número finito de movimentos.
- Discreto: Jogadores alternam movimentos, sem movimentos simultâneos.
- Determinístico: Sem aleatoriedade.
- Multi Jogadores: Pelo menos dois jogadores se enfrentando.
- Informação Completa: Todas as informações são conhecidas por todos os jogadores, não possui elementos ocultos.

Por conta dessas características, todos os jogos pertencentes a essa classe são representáveis como uma máquina de estados finita (LOVE *et al.*, 2008) e GDL foi desenvolvida como uma representação compacta da máquina de estados.

Contudo, a GDL de Stanford não permite que se descreva jogos com informação incompleta e com aleatoriedade. Para contornar essa dificuldade, (THIELSCHER, 2011a) estendeu a linguagem para GDL-II, GDL com informação incompleta, que permite a modelagem de jogos que possuem elementos de aleatoriedade, como lançamento de dados e compra de cartas de um deck, assim como jogos com informação incompleta, como por exemplo poker. Assim, GDL-II é capaz de descrever uma quantidade ainda maior de jogos. Segundo (THIELSCHER, 2011b), todo jogo da classe pode ser descrito a partir dessa extensão da Stanford GDL.

Também foi desenvolvida a GDL-III (THIELSCHER, 2016), GDL de informação imperfeita e de introspecção, que permite, além de descrever todos os jogos que a GDL-II descreve, a modelagem de jogos epistêmicos. Jogos epistêmicos são jogos onde o conhecimento que os jogadores possuem sobre os estados de todos os jogadores é relevante para a o objetivo do jogo.

Essas três linguagens são muito expressivas, permitindo que uma grande gama de jogos seja descrita nelas, e todas as extensões da GDL visam aumentar sua expressividade. Contudo, por ser pouco específica, as descrições geradas através da GDL

e suas variantes são muito verbosas (MAHLMANN *et al.*, 2011). Ou seja, um jogo bem simples demanda uma descrição muito extensa, de difícil leitura, demandando por exemplo três páginas de texto para descrever um jogo da velha.

2.4 Descrições de jogos mais específicos

Uma abordagem interessante para contornar o problema da verbosidade da Stanford GDL foi de desenvolver outras linguagens de descrição, mais específicas a determinados domínios de jogos, permitindo que jogos sejam descritos de forma mais concisa.

A SGDL, *Strategy Game Description Language* (MAHLMANN *et al.*, 2011), foi cunhada com o intuito de descrever jogos de estratégia e faz uso de uma representação baseada em árvore para descrever as mecânicas. Em sua tese, BROWNE (2008) criou a Ludi GDL, que descreve jogos combinatoriais de dois jogadores. As regras do jogo são representados como árvores de expressão para facilitar o processo evolucionário para criação de novas regras. A linguagem de descrição VGDL (EBNER *et al.*, 2013), criada para a descrição da classe de jogos de videogame, define as entidades e interações que podem ocorrer durante o jogo a partir de uma estrutura de árvore.

Em relação ao domínio de jogos de cartas, abrangendo um gênero ainda mais específico, CORREIA (2013) concebeu uma linguagem para descrição de variantes do jogo de Poker, com uma estrutura baseada em XML.

Um trabalho interessante foi realizado por FONT *et al.* (2013) ao criar uma linguagem de descrições de jogos de carta, a partir de uma gramática livre de contexto, $G_{cardgame}$. Essa linguagem foi criada exclusivamente com o objetivo de gerar aleatoriamente jogos de cartas jogáveis e, por consequência, é considerada limitada em relação a quantidade de jogos de cartas que descreve. Dentre algumas das limitações da linguagem, o trabalho de BELL & GOADRICH (2016) cita:

- A impossibilidade de voltar em um estágio anterior.
- A limitação em utilizar somente o *deck* francês de 52 cartas, não permitindo jogos com cartas faltantes ou excedentes, como por exemplo Buraco em que se joga com dois baralhos de 52 cartas.
- Cada jogador possui somente uma mão.
- Com exceção ao *deck*, as localizações de cartas na mesa são todas com cartas reveladas, impossibilitando mecânicas de esconder cartas dos oponentes na mesa.
- Não existe uma separação clara entre ações do jogador e controle de fluxo, permitindo que um jogador decida quando um estágio do jogo termina.

- A precedência de cartas é fixada no começo do jogo, não permitindo troca de ranking das cartas.

- A ordem dos jogadores é fixada, não sendo possível alterar a ordem de turno.

2.5 Importância de modelar Dados

Jogos de dados são muito populares em várias partes do mundo. Dados adicionam um fator interessante de aleatoriedade nos jogos. Alguns jogos resolvem as disputas entre jogadores de forma determinísticas enquanto outros utilizam de dados para solucionar esse confronto, implicando em uma incerteza prévia em relação ao vencedor. Dados possuem diferentes quantidades de lados, os mais comuns sendo 6 lados e 20 lados.

Ao modelar as regras e a aleatoriedade gerada pelo dado, um designer de jogos pode escolher a melhor combinação de quantidade de dados e lados, para obter o melhor design de jogo.

Segundo ISAKSEN *et al.* (2016), a modelagem de dados pode ser usada para os seguintes objetivos:

- Um designer pode desejar um alto grau de incerteza em seu jogo, aumentando os riscos que os jogadores precisam tomar para vencer.

- Um designer pode querer diminuir a disparidade de pontos entre jogadores, não permitindo que um jogador se distancie dos demais em demasia, usando a aversão à iniquidade para criar jogos mais balanceados. Descrita por economistas como o desejo de minimizar a desigualdade, a aversão à iniquidade define que pessoas preferem que as recompensas sejam distribuídas igualmente ISAKSEN *et al.* (2015). O design de jogos com dados permite que esta estratégia seja implementada, gerando uma experiência mais competitiva.

- A inserção de aleatoriedade permite que um jogo aparente ser mais balanceado, visto que o jogador mais fraco estrategicamente pode vencer do jogador mais forte.

- A impossibilidade em dominar a incerteza pode ser um fator interessante como apelo do jogo.

- O designer pode ajustar a aleatoriedade para incentivar situações apropriadas para o jogo.

2.6 RECYCLE

RECYCLE é um linguagem de descrição de jogos de cartas desenvolvida por (BELL & GOADRICH, 2016). Seu nome vem de *REcursive CYclic Card game LanguagE*, ou seja linguagem de descrição de cartas recursiva e cíclica, e significa que a linguagem permite a descrição de jogos de cartas com recursão de etapas, cada uma delas contendo ciclos de movimentos dos jogadores. Por se assemelhar à linguagem de programação LISP, uma grande quantidade de instruções de controle do fluxo do jogo são aninhadas, indicando elementos de informação, fluxo de controle ou execução condicional.

RECYCLE permite modelar e descrever jogos de cartas. Em contrapartida à *Stanford GDL*, jogos de cartas possuem aleatoriedade e são de informação incompleta, com cartas sendo ocultas de outros jogadores. Algumas das vantagens do RECYCLE em relação ao $G_{cardgame}$ são:

- Ciclos de controle de fluxo, a partir de regras com estrutura de repetição.
- Não se limitar ao baralho francês de 52 cartas, permitindo que o designer crie novas combinações de cartas.
- Cada jogador pode controlar diversas localizações de cartas.
- Permite localizações ocultas além das mãos.
- O jogador é limitado a realizar suas possíveis ações e caso não haja ação possível, seu turno termina automaticamente.
- O ranking das cartas pode ser atualizado de acordo com a dinâmica do jogo..
- A ordem do jogo é cíclica, podendo definir a ordem dos jogadores de acordo com a pontuação da última rodada.
- Possui indicadores de fichas, útil para jogos como Poker onde fichas são essenciais.

Para criar um jogo completo, é necessário seguir uma estrutura padrão. A figura 2.3 mostra os elementos básicos para a criação de um jogo, a partir da regra de produção *game*.

A primeira regra *declare* consiste na declaração de variáveis globais.

Em seguida, define-se um bloco chamado *setup*, responsável pela criação dos componentes que serão usados no jogo: quantidade de jogadores, times dos jogadores, criação de diferentes cartas e suas quantidades.

Posteriormente, ações ou estágios podem ser chamados indiscriminadamente.

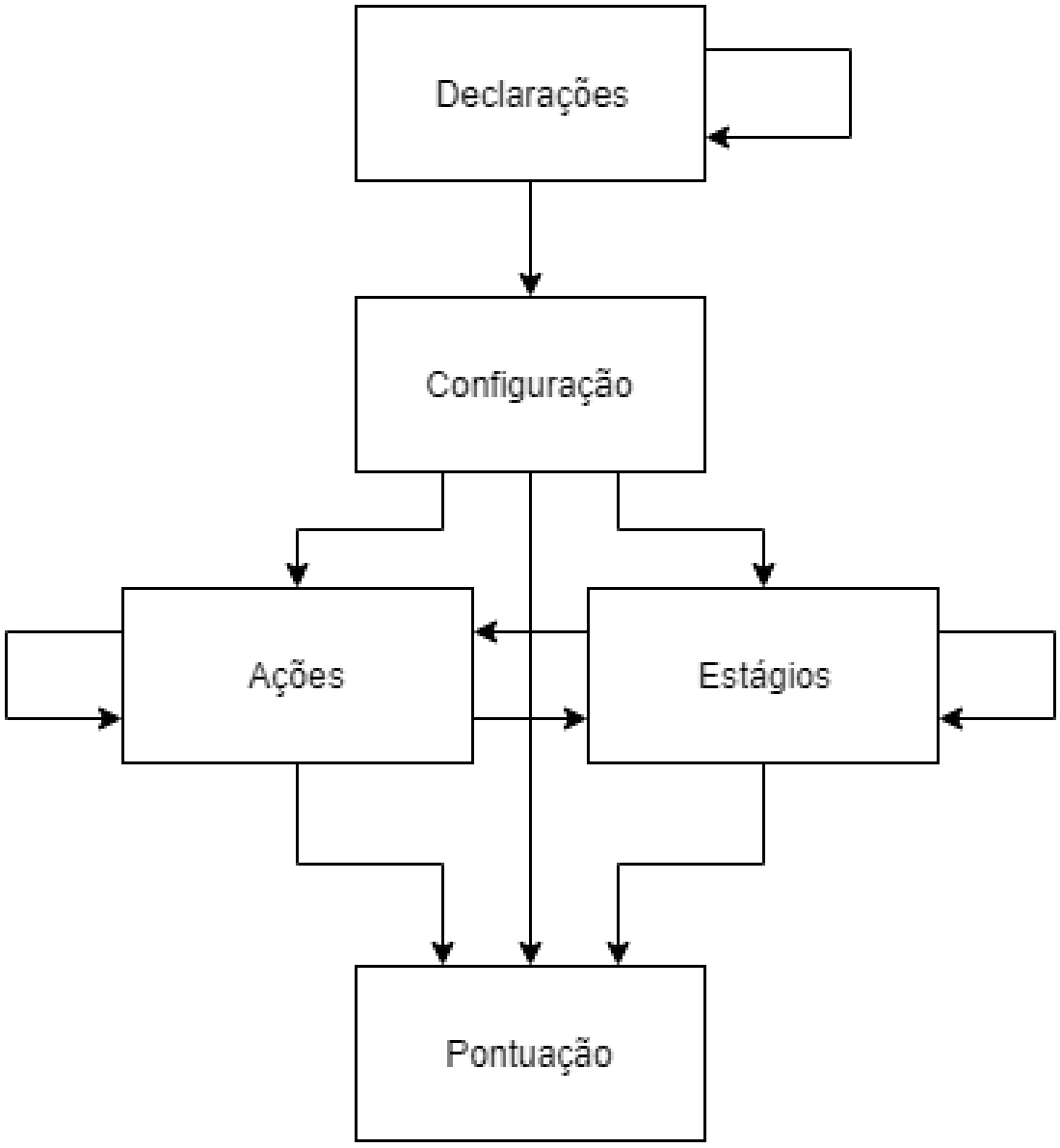

Figura 2.3 – Estrutura de Criação de um Jogo em RECYCLE

Ações correspondem tanto às ações realizadas pelo computador, simulando um *dealer*, quanto a movimentos específicos de jogadores. Assim, exemplos de ação são: embaralhamento das cartas, distribuição de cartas, remoção automática de cartas, um movimento específico de um determinado jogador. Elas correspondem à regra *multiaction* da gramática.

Estágios são o bloco fundamental de fluxo de controle do RECYCLE. Ele atua iterando sobre todos os jogadores, seguindo um ciclo, até atingir a condição de parada. Estágios são definidos através da regra *stage* na gramática. Enquanto a condição de parada de um estágio, que corresponde a uma expressão booleana testando um estado do jogo, não for atingida, o estágio continua sendo executado, passando a vez para o próximo jogador. É no estágio que os jogadores realizam seus

movimentos e é a regra que mais se assemelha à definição de uma rodada. Estágios podem conter outros estágios, formando assim estágios aninhados.

Como último estágio da descrição de um jogo, a pontuação final é verificada no bloco definido pela regra *scoring*, que realiza a validação dos pontos iterando sobre cada um dos jogadores e, de acordo com uma variável de pontuação, define o vencedor como aquele que possui a melhor pontuação.

Todos esses elementos são descritos a partir da gramática do RECYCLE, que a imagem 2.4 explicita o diagrama de sintaxe.

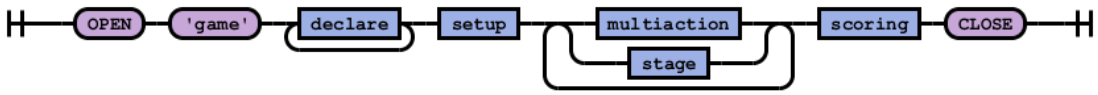

Figura 2.4 – Diagrama de Sintaxe da Regra *game*

2.6.1 Descrição de um jogo em RECYCLE

Para exemplificar a linguagem, esta seção apresenta um jogo descrito através de Recycle. Agram corresponde a um jogo de cartas de vaza de quatro jogadores. Cada jogador recebe 6 cartas de um baralho padrão, com o ás de espadas sendo removido do *deck*. O jogador vence uma rodada de mãos caso ele jogue a carta mais alta, seguindo o naipe do turno. Ele começa jogando no próximo turno. O vencedor do jogo é dado pelo jogador que vence a última rodada.

O apêndice A.1 apresenta o código A.1, referente à descrição do jogo na linguagem RECYCLE, assim como descreve uma explicação linha a linha do código.

2.7 Interpretando uma Linguagem de Descrição

A linguagem RECYCLE foi definida em uma gramática livre de contexto. Para poder criar um interpretador capaz de ler e entender uma descrição implementada nesta linguagem, uma estrutura reconhecedora da linguagem é necessária.

O ANTLR, **AN**other **T**ool for **L**anguage ***R***ecognition, é um gerador de analisador sintático para leitura, processamento, execução ou tradução de texto estruturado (PARR, 2013). A partir de uma gramática, essa ferramenta é capaz de construir um analisador sintático que, por sua vez, constrói árvores de análise sintática.

```
soma : digito + digito
digito : 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
```

Código 2.1 – Exemplo de gramática de soma de dois dígitos

2.7.1 Gramática Livre de Contexto

Uma gramática livre de contexto corresponde na especificação da sintaxe de uma linguagem. A gramática constitui um conjunto de regras de produção que especificam a estrutura da linguagem.

Segundo (AHO, 2003), uma gramática livre de contexto é definida por quatro componentes:

1. Tokens, também chamados de símbolos terminais.
2. Símbolos não-terminais.
3. Produções, sendo compostas por um não-terminal do lado esquerdo da produção e uma sequência de tokens e/ou não-terminais do lado direito da produção.
4. Um símbolo de partida, indicando que um não-terminal corresponde à regra inicial da gramática.

O exemplo no código 2.1 apresenta duas regras de produção, a primeira com o símbolo não-terminal *soma* do lado esquerda e com os símbolos não-terminais *digito* separados pelo símbolo terminal +. A segunda regra apresenta o símbolo não-terminal *digito* do lado esquerdo, e os símbolos terminais representando os dígitos do lado direito. O símbolo *soma* corresponde ao símbolo de partida, visto que suas produções aparecem em primeiro.

2.7.2 Analisador Sintático

Duas etapas são necessárias para a análise de uma descrição, a análise léxica e a análise sintática:

A análise léxica é o processo responsável por analisar uma gramática e agrupar os caracteres em palavras ou símbolos, *tokens*. O programa que executa a análise léxica é chamado de analisador léxico. Ele reconhece assim as palavras reservadas, variáveis, dentre outros, e desta forma identifica os diferentes *tokens*.

A partir da descrição formal da linguagem, a gramática, analisadores sintáticos são construídos. Um analisador sintático corresponde a um programa capaz de reconhecer uma linguagem. Ele é responsável por verificar se um texto descrito a partir da gramática está correto sintaticamente, comparando a estrutura da frase

com as regras da gramática. O analisador sintático constrói uma árvore de análise sintática do texto recebido como entrada.

ANTLR gera Analisadores Sintático Descendentes, que fazem uso da estratégia de derivação: procurando chegar à sentença a partir do símbolo inicial da gramática e derivando de forma descendente as regras.

2.7.3 Árvore de Análise Sintática

Uma árvore sintática representa a estrutura sintática da cadeia de acordo com uma gramática livre de contexto. A raiz da árvore corresponde ao símbolo de partida, os nós internos correspondem aos símbolos não-terminais da gramática e as folhas da árvore são os nós terminais da gramática.

A figura 2.5 apresenta o fluxo de criação da árvore de análise sintática a partir de uma sentença. O analisador léxico divide a sentença em *tokens*, reconhecendo cada um deles. Posteriormente, o analisador sintático reconhece a estrutura dos *tokens*, criando a árvore de análise sintática de acordo com a gramática.

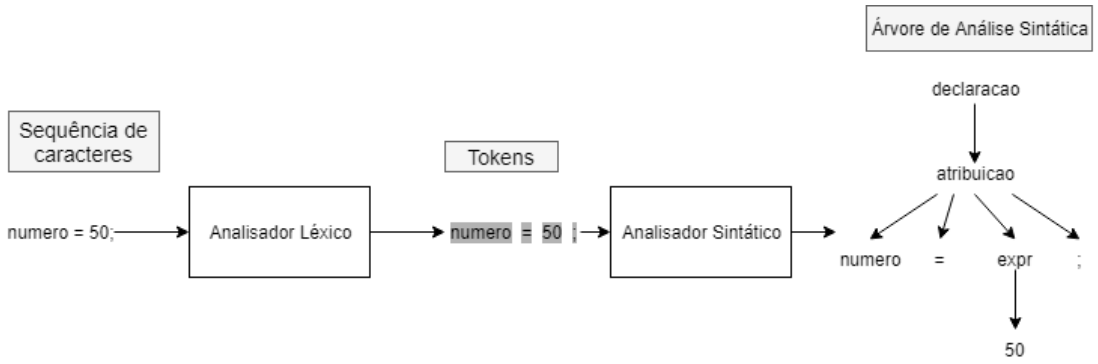

Figura 2.5 – Fluxo de criação da árvore de análise sintática, baseado em (PARR, 2013)

2.7.4 Navegando a árvore

Com o objetivo de criar uma aplicação capaz de interpretar a linguagem, é necessário estabelecer os comportamentos de cada regra. Assim, para cada regra, o interpretador precisa executar um método apropriado com a ação desejada. Desta forma, para percorrer a árvore, precisa-se implementar e executar a travessia da árvore, que chama os métodos correspondentes a cada nó.

2.8 CARDSTOCK

CARDSTOCK é um simulador de jogos da linguagem RECYCLE. Implementado em C# por (BELL & GOADRICH, 2016), esse simulador é capaz de receber uma

descrição de um jogo de cartas descrito a partir da gramática do RECYCLE e a partir disso simular sua execução com diferentes jogadores.

Para realizar essas simulações, o CARDSTOCK comporta a *CardEngine*, um modelo geral do tabuleiro responsável por instanciar os componentes do tabuleiro, assim como realizar as possíveis ações. Esse modelo comporta uma classe *CardGame* que simula o tabuleiro com instâncias de classes representando jogadores, cartas, localizações, fichas de pontos, ordem de jogo, dentre outros.

O CARDSTOCK realiza a implementação dos métodos chamados ao realizar a travessia da árvore de análise sintática gerada pelo ANTLR, especificando o comportamento desejado ao visitar cada nó da árvore. O trabalho de BELL & GOADRICH (2016) nomeou esse componente como iterador de árvore. Para melhor compreensão, o termo **interpretador** será utilizado nesta dissertação para designar os métodos que implementam o comportamento da gramática e realizam a travessia da árvore,

Outro componente essencial no CARDSTOCK é a capacidade de adicionar agentes diferentes. Assim, a simulação independe de um jogador humano, sendo possível fazer uso de qualquer jogador que se desejar. Desta forma, diferentes agentes inteligentes podem ser inseridos na ferramenta conforme o perfil de simulação desejado.

Como mencionado anteriormente, a competição de General Game Playing foi importante para a criação de novos agentes inteligentes capazes de jogar uma grande quantidade de jogos nunca vistos antes por eles. Dentre várias técnicas pesquisadas, o algoritmo que obteve o melhor desempenho por anos foi o de Árvore de Busca Monte-Carlo (MCTS), como visto nos vencedores da competição nos anos de 2007 e 2008 (BJORNSSON & FINNSSON, 2009). Efetivamente, para jogar eficientemente jogos genéricos, técnicas baseadas em Monte-Carlo emergiram nos últimos anos (CHASLOT *et al.*, 2008).

2.9 Mecânicas de Jogos de Tabuleiro

Esta seção apresenta a definição algumas das mecânicas essenciais para um jogo de cartas e dados, utilizadas no trabalho.

(MANGELI, 2016) definiu jogos **baseados em turnos** como aqueles nos quais as partidas são compostas por turno, e em cada turno, todos os jogadores fazem seus movimentos. Segundo o mesmo autor, o conceito de **movimento** consiste na escolha de ações do jogador. Assim, os movimentos do jogadores, realizados dentro de um turno, podem ser simultâneos ou um de cada vez. Também é importante definir que o termo **rodada** é um sinônimo de turno e que ambos os termos são usados neste trabalho com o mesmo sentido.

Em sua Ontologia de Mecânicas de jogos de tabuleiro, (KRITZ *et al.*, 2017) definiu elementos de representação de informação a partir de componentes que ar-

mazenam e transmitem os elementos do jogo. Dentre eles, o **Dado** como objeto é uma mecânica útil para geração de número aleatórios e como marcador de pontos. Um dado possui como característica elementar sua quantidade de faces, possuindo pelo menos duas. Diz-se que um dado é justo se a probabilidade de cair em uma face é igual para todas as faces.

Dentre as mecânicas de ação, KRITZ *et al.* (2017) cita os mecanismos para geração de objetos ou valores aleatórios. **Lançamento de um dado** é uma mecânica onde o jogador rola um dado de qualquer tipo e recupera o valor da face que caiu para o alto, com o intuito de gerar aleatoriedade e criando resultados inesperados no jogo. A notação padrão para caracterizar o lançamento de um dado é a quantidade de faces de um dado precedido pela letra **d**. O lançamento de um dado padrão de seis faces é representado como **d6**.

O conceito de **Área** é definido por KRITZ *et al.* (2017) como uma mecânica de representação de informação, onde o espaço de jogo é dividido em áreas que possuem outros componentes. Assim, uma área pode possuir um conjunto de dados. A notação para o lançamento de múltiplos dados corresponde à notação do lançamento de um dado precedido da quantidade de dados lançados. Assim, o lançamento de três dados de seis faces é representado como **3d6**. Múltiplos dados de tamanhos diferentes podem ser representados através da notação de soma, como por exemplo a soma de três dados de seis faces com 2 dados de 3 faces, **3d6 + 2d3**.

(KRITZ *et al.*, 2017) também define **Carta** como um elemento de representação de informação. Sendo assim, Carta é uma mecânica para o uso de cartas em um jogo, um componente versátil que pode ir além do baralho tradicional, sendo usado como gerador de eventos aleatórios, limitante de recursos, dentre outros. Em relação à mecânica de ação que gera valores aleatórios, **Deck** consiste no mecanismo onde um grupo de cartas é agrupado e embaralhado, e jogadores selecionam a carta do topo, não sabendo qual carta foi selecionada.

2.10 Métricas de Estética

Medir a estética de um jogo corresponde a um desafio. Avaliar o sentimento de uma pessoa ao jogar é uma questão muito subjetiva e diferentes pessoas podem ter diferentes sensações. Diversos autores criaram e sugeriram critérios para quantificar a estética, como abordado mais adiante. As partir destes critérios, métricas podem ser desenvolvidas. Uma métrica consiste em uma forma de avaliar e medir determinados critérios em relação a um jogo.

THOMPSON (2000) definiu quatro critérios de qualidade que um jogo precisa possuir para ser interessante:

- Profundidade: Capaz de ser jogado em diversos níveis de dificuldade.

- Clareza: Um jogador com um conhecimento não profissional do jogo é capaz de entender o melhor movimento para a situação.
- Drama: Se um jogador é capaz de conseguir se recuperar de uma posição mais fraca e ganhar o jogo.
- Conclusivo: Deve ser possível que o jogador na frente assuma uma vantagem tal que o outro jogador não seja capaz de se recuperar.

Essas quatro qualidades foram definidas para jogos abstratos de estratégia, com informação perfeita e dois jogadores.

Para agilizar o processo de criação de novos jogos, ALTHÖFER (2003) propôs um conjunto de critérios para medir o quão interessante um jogo é. Com o intuito de gerar jogos com a ajuda de computadores, essas métricas foram criadas considerando somente aspectos avaliados automaticamente, a partir da análise estatística dos registros. Desta forma, concebeu-se que determinados critérios independem do perfil do jogador, são qualidades intrínsecas do jogar.

BROWNE (2008) desenvolveu um gerador automático de jogos cuja classe de jogos consiste em combinatórios, entre dois jogadores, soma zero e discretos. Baseado nessa restrição, ele criou um conjunto extenso de critérios e métricas de estética.

2.10.1 Framework de avaliação de Mangeli

Em sua dissertação de mestrado, MANGELI (2016) desenvolveu uma ferramenta matemática e estatística que permite analisar uma série de amostras de eventos para inferir o comportamento dessa população.

A partir das métricas concebidas por BROWNE (2008), MANGELI (2016) adaptou e criou seis métricas de estética, com o objetivo de aplicá-las a jogos de vários jogadores baseados em turno. Suas métricas foram aplicadas a conjuntos de dados do desafio Sebrae e do Brasileirão. Essas métricas são:

- Drama por Pontos
- Drama por Posição
- Drama por Caminho
- Mudança de Líder
- Incerteza por Entropia
- Incerteza por PDD

2.10.2 Drama

O critério estético de Drama consiste em avaliar o quão dramática foi uma partida. Se o Drama for baixo, significa que o vencedor esteve na liderança durante muitas rodadas. Se o Drama for alto, significa que, durante muitas rodadas, o vencedor não estava liderando.

2.10.2.1 Drama por Pontos

O Drama por Pontos de uma partida é definido pela diferença de pontos entre o jogador que venceu e o eventual líder nos turnos em que o vencedor não estava liderando.

A métrica é apresentada em forma de equação a seguir. S_H corresponde à maior pontuação da partida, atingida pelo vencedor, e S_L o limite mínimo da pontuação. O denominador dentro da raiz é necessário para uma normalização. Cada movimento é chamado de m. Em m_n estão sendo considerados somente os turnos em que o vencedor não estava liderando. Desta forma, soma-se a diferença normalizada de pontos a cada rodada n entre a pontuação do líder, $S_l(m)$, e o jogador que venceu o jogo, $S_w(m)$. Por fim, esse valor é dividido pela quantidade de rodadas em que a pontuação do vencedor foi menor que a do líder.

$$Drama\,por\,Pontos_{mch} = \frac{\sum_{n=1}^{|M|-1} \sqrt{\frac{S_l(m_n)-S_w(m_n)}{(S_H-S_L)}}}{|\{m|(S_w(m) < S_l(m))\}|} \tag{2.1}$$

2.10.2.2 Drama por Posição

O Drama por Posição corresponde à soma das distâncias em posições do jogador que venceu e o eventual líder nos turnos em que o vencedor não estava liderando. A partir de um Vetor de Classificação de Jogadores PRV_m, que classifica os jogadores a cada rodada, e de uma função de posição P_f, calcula-se a distância em posição entre o jogador que venceu e a primeira posição. Assim, soma-se a diferença de posições normalizada em todas as n rodadas em que o jogador vitorioso P_w não estava liderando.

$$Drama\,por\,Posicao_{mch} = \frac{\sum_{n=1}^{|M|-1} \sqrt{\frac{P_f(P_w,m_n)-1}{|P|-1}}}{|\{m|(P_f(P_w,m) > 1)\}|} \tag{2.2}$$

2.10.2.3 Drama por Caminho

O Drama por Caminho representa o quão similar o caminho percorrido pelo jogador vencedor foi em relação ao Caminho de Drama Máximo, MDP. Segundo MANGELI

(2016), define-se Caminho de Drama Máximo como o caminho mais longo que um jogador percorre saindo da última posição e sendo o primeiro ao final do jogo. O MDP de um turno m é dado pela quantidade de jogadores P, representando as posições à serem ultrapassadas, somado aos turnos anteriores em que o jogador permaneceu naquela posição. O MDP pode ser visto como a diagonal de drama máximo.

$$MDP(m) = \left\lceil |P| + \frac{(1 - |P|)(m-1)}{|M| - 1} \right\rceil \tag{2.3}$$

O Drama por Caminho deve ser calculado de acordo com a similaridade entre o caminho percorrido pelo vencedor e o Caminho de Drama Máximo. Para calcular essa métrica, compara-se a posição do jogador vencedor na rodada com a posição correspondente no MDP. Se ambos estão na mesma posição, significa que, para aquela determinada rodada, o jogador está no Caminho de Drama Máximo. A equação considera somente rodadas onde o jogador vencedor não era o jogador na liderança.

$$Drama\,por\,Caminho_{mch} = \frac{|\{m|P_f(P_w, m) > 1)\}|}{|M| - 1} \left(1 - \sum_{m=1}^{M} \frac{|P_f(P_w, m) - MDP(m)|}{(|P| - 1)(|M| - 1)}\right) \tag{2.4}$$

2.10.3 Mudança de Liderança

A mudança de liderança é um critério importante e desejado em um jogo. Um jogo em que não ocorre mudança de liderança corresponde a um jogo menos interessante do que um jogo em que ocorre uma ou mais mudança de liderança. Sendo L o conjunto de jogadores que lideraram durante o jogo e e $LChange$ o conjunto de turnos em que ocorreu uma mudança de liderança, a fórmula da mudança de liderança é apresentada como a média aritmética da soma do número de líderes com o número de mudanças, normalizada entre zero e um.

$$Mudança\,de\,Liderança_{mch} = \frac{\sqrt{\frac{|L|-1}{|P|-1}} + \sqrt{\frac{|LChange|}{|M|-1}}}{2} \tag{2.5}$$

2.10.4 Incerteza

Incerteza é um aspecto fundamental do jogo. Se um jogo começasse com um vencedor definido, esse jogo não possuiria apelo nenhum. Desta forma, não saber de antemão quem será o vencedor implica em uma sensação de dúvida e incerteza, dois sentimentos essenciais para um bom jogo. Essa incerteza dura até uma determinada rodada, na qual, a partir dela, o vencedor já está bem definido pelas regras do jogo.

A incerteza abordada por MANGELI (2016) se refere à incerteza em relação ao final do jogo, em não saber quem vence a partida e quanto tempo dura essa dúvida. A incerteza máxima ocorre quando todos os jogadores possuem a mesma probabilidade de vencer a partida. As duas métricas a seguir calculam o mesmo critério, fazendo uso de fórmulas diferentes para seu cálculo.

2.10.4.1 Incerteza por entropia

Após uma rodada m, temos $\mathbb{P}(p, m)$, que corresponde à probabilidade do jogador p vencer a partida após a rodada m ter sido avaliada. Então, dada uma rodada m, a soma da probabilidade de cada jogador vencer corresponde a 1.

A entropia corresponde a uma forma de medir o grau de incerteza em relação a um determinado evento. Assim, a entropia aumenta de acordo com a incerteza em relação ao resultado de um experimento. MANGELI (2016) usou do valor máximo da entropia de Shannon para chegar a essa equação.

A equação correspondente à incerteza por entropia na partida realiza a média da incerteza em cada um dos turnos, excluindo o último turno.

$$Incerteza\ por\ Entropia(m) = \frac{-\sum\limits_{p \in P} \mathbb{P}(p, m) \log_2(\mathbb{P}(p, m))}{\log_2(|P|)} \tag{2.6}$$

2.10.4.2 Incerteza por distância de distribuição de probabilidade

Uma outra métrica para calculo da incerteza corresponde a definir a distribuição de probabilidade de vitória de cada jogador. Desta forma, se $D = (1, 0, 0, \ldots, 0)$, tem-se ausência de incerteza. A partir da distância entre distribuições, foi-se capaz de derivar a seguinte métrica, que fez uso de Distância Hellinger.

$$Incerteza\ por\ PDD_{mch} = \frac{\sum\limits_{n=1}^{|M|-1} 1 - \sqrt{\sum\limits_{p \in P} \frac{\left(\sqrt{\mathbb{P}(p, m_n)} - \frac{1}{\sqrt{|P|}}\right)^2}{2 - \frac{2}{\sqrt{|P|}}}}}{|M| - 1} \tag{2.7}$$

2.11 Trabalhos Relacionados

Como citado anteriormente, BROWNE (2008) desenvolveu a Ludi GDL e um gerador automático de jogos para uma classe específica de jogos. Também cunhou uma extensa lista de métricas, em sua maioria baseadas em simulação. A partir dessas métricas, ele gerou automaticamente novos jogos, dentro os quais um se torna um sucesso comercial.

CORREIA (2013) desenvolveu uma linguagem de jogos de poker e realizou *playtests* com pessoas para avaliar seu funcionamento.

FONT *et al.* (2013) desenvolveu uma linguagem de jogos de cartas, gerou variantes de forma automatizada e simulou jogos com jogadores aleatórios, recuperando os critérios de avaliação: frequência de vitória para cada jogador, número de empates, número de jogos que ocorreram erro e média da quantidade de turnos necessários para fim do jogo.

ISAKSEN *et al.* (2016) cunhou a métrica de proximidade, além de ter adaptado as métricas de viés de vitória e porcentagem de empates. Seu trabalho emprega essas métricas para avaliar o comportamento de jogos de dados ao variar a combinação de dados.

BELL & GOADRICH (2016) descreveu uma linguagem de jogos de cartas e um simulador que serviram como base de desenvolvimento para esse trabalho. Também avaliou seu jogos, mas com critérios de características do jogo, tais como fator de ramificação de decisões, percentual de vitória para cada jogador, duração do jogo, vantagem de turno e potencial para estratégia.

Capítulo 3

Proposta

Esse capítulo apresenta as ideias propostas nessa dissertação. Assim, ele expõe uma proposta de uma extensão da linguagem de jogos de cartas RECYCLE, adicionando o elemento de dados e permitindo a descrição de jogos de cartas que possuem dado, denominada RECYCLEDICE. Também foi modificado o sistema CARDSTOCK, cuja extensão denominou-se de GAMESTOCK. Além disso, esse trabalho apresenta a modelagem da adaptação do simulador e da gramática para que a pontuação a cada turno fosse armazenada e exportada. Essa dissertação também expõe a adequação do modelo de jogo do *framework* de avaliação de Mangeli (MANGELI, 2016) para permitir jogos genéricos no formato necessário. Por fim, a ferramenta de avaliação foi adaptada para receber uma grande quantidade de jogos e, por sua vez, foi avaliada em relação às métricas.

Para resolver o problema da pesquisa, foi necessário modificar a modelagem da gramática da linguagem, e, por consequência, reescrever o interpretador da árvore de análise sintática. Além disso, com o objetivo de mostrar o funcionamento da modificação na linguagem, descrições de jogos fazendo uso da extensão precisaram ser detalhadas. A partir da descrição e de simulações executadas, um novo modelo de jogo precisou ser especificado na ferramenta de avaliação, que permitisse a avaliação de jogos no formato desejado. Finalmente, este novo modelo foi modificado de forma a aceitar e calcular as métricas de todas as simulações geradas.

3.1 Proposta Geral do Problema

O problema de avaliação automatizada de jogos pertence a um problema maior, a geração automatizada de jogos. Para poder criar novos artefatos lúdicos automaticamente, é necessário uma estrutura eficaz de avaliação de jogos, que considere todas as etapas.

A imagem 3.1 demonstra as diferentes etapas propostas neste projeto. Uma **gramática** foi desenvolvida e descrita, permitindo especificar a linguagem RECYCLE-

DICE. A partir desta linguagem, um **analisador sintático** foi gerado. Ao receber uma **descrição de jogo** especificada a partir da RECYCLEDICE, o analisador sintático produz uma **árvore de análise sintática**. Cada descrição reconhecida pela gramática gera uma árvore de análise sintática diferente. Para cada regra do analisador sintático, um **interpretador** no sistema GAMESTOCK ficou responsável por executar o código correspondente. Paralelamente, a modelagem do jogo, renomeada como ***Game Engine***, foi realizada no sistema. O GAMESTOCK interpretou a árvore de análise sintática, construiu as instâncias no modelo e os ***players*** ficaram responsáveis por jogar e interagir com o modelo. Assim, as **simulações** puderam ser realizadas e estatísticas de cada iteração foram extraídas. Elas serviram como entrada para a ***Framework* de Avaliação de Mangeli**, onde **métricas de estética** foram implementadas. Por fim, a ferramenta gerou os **dados de avaliação de estética**.

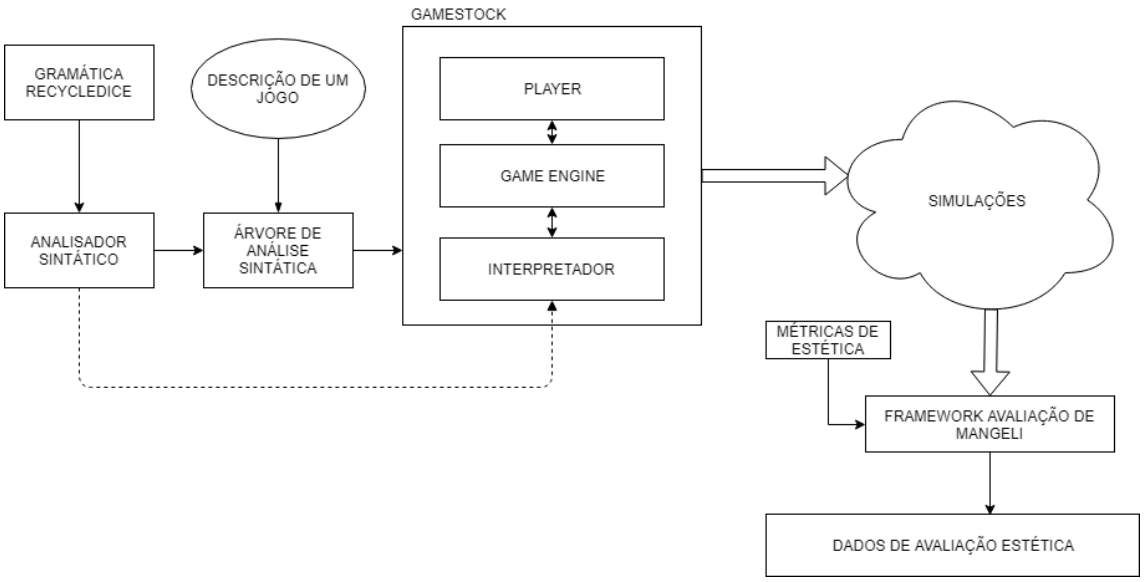

Figura 3.1 – Etapas para Avaliação Automatizada

Esta arquitetura foi escolhida e desenvolvida com base em alguns fatores importantes. Primeiramente, como elemento essencial do sistema, fez-se a escolha por uma linguagem de descrição de jogos. Essa linguagem deveria ser capaz de descrever uma grande quantidade de jogos, assim como a *Stanford GDL*, contudo não sendo tão generalista. A verbosidade sendo desta forma um fator importante para a escolha da linguagem. Para contrapor a questão da verbosidade, linguagens de domínio mais específicos foram pesquisadas. Com base nas diferentes linguagens estudadas e analisadas, optou-se por estender a linguagem RECYCLE (BELL & GOADRICH, 2016). Por ser um componente interessante em jogos de tabuleiros, foi adicionada a mecânica de lançamento de dados na linguagem. Assim, a linguagem de descrição RECYCLEDICE serviu como linguagem de base.

Outro elemento importante nesta arquitetura corresponde ao GAMESTOCK. Dada a necessidade de interpretar e simular os jogos com dados, o GAMESTOCK foi desenvolvido para cumprir esse papel. Ao receber uma árvore de análise sintática correspondente a uma descrição de jogos de cartas, o interpretador itera sobre os nós da árvore e avalia cada declaração para executar o código específico. Desta forma, as ações correspondentes são realizadas na *Game Engine*, simulando o jogo. A escolha deste modelo se deu pela adaptação da ferramenta CARDSTOCK, cuja estrutura serviu como base.

Ao *player*, que corresponde a um agente, são apresentadas ações possíveis, à medida do decorrer do jogo, a partir da *Game Engine*. Neste trabalho, para fazer a validação da estrutura, foi utilizado jogadores com escolhas randômicas. O componente de jogador consiste em um fator importante na avaliação de estética, mudança de jogadores acarreta em diferentes resultados no jogo que por sua vez implica em estéticas distintas.

Com as simulações executadas, muitas métricas podem ser utilizadas e desenvolvidas. Devido o interesse por realizar uma avaliação estética dos jogos, ao mesmo tempo que jogos de cartas e dados correspondem a uma grande gama de jogos de turno, optou-se pelo uso do *Framework* de Avaliação de Mangeli. Este, por sua vez, foi adaptado para permitir as simulações como entrada e devolver as avaliações de estética do conjunto de simulações como saída.

3.2 Modelando dados

Este trabalho fez uso como base a linguagem RECYCLE e o simulador CardStock. Como visto na seção 2.6, a linguagem de descrição de jogos RECYCLE consegue descrever uma gama importante de jogos de cartas. Contudo, desejou-se estender a linguagem com o intuito de adicionar o componente de dado. Essa extensão permitiu que alguns objetivos fossem atingidos:

- Possibilitou que jogos existentes de cartas com dados sejam modelados.
- Viabilizou a adição do componente de dado em jogos de cartas existentes, permitindo desta forma variantes com os dois elementos.
- Proporcionou a possibilidade de que uma classe de jogos exclusivamente de dados seja modelada.

Numa primeira vista, modelar um jogo que só possui dados em um linguagem criada prioritariamente para cartas não possui fundamento. Contudo, como jogos com ambos os componentes podem ser modelados e a estrutura já existe na gramá-

tica, fez-se proveito disso, sem precisar reescrever toda uma gramática, a modelagem do jogo e o interpretador.

Como já visto na seção 2.5, existe uma motivação importante para o uso desse componente em novos jogos desenvolvidos por um designer.

Para poder modificar e reescrever a gramática, dois outros componentes do simulador precisaram ser atualizados. Eles se relacionam de forma que mudanças em cada um implicam em atualização dos outros dois, como mostra a figura 3.2. Assim sendo, a proposta de extensão da linguagem foi realizada com base em três pilares:

1. Adição de regras na gramática de forma a permitir que novos jogos fossem descritos.

2. Desenvolvimento das novas classes na *Game Engine*, que corresponde à estrutura de dados onde o estado do jogo é carregado.

3. Por fim, a implementação do comportamento do interpretador nos novos métodos necessários para travessia da árvore de análise sintática.

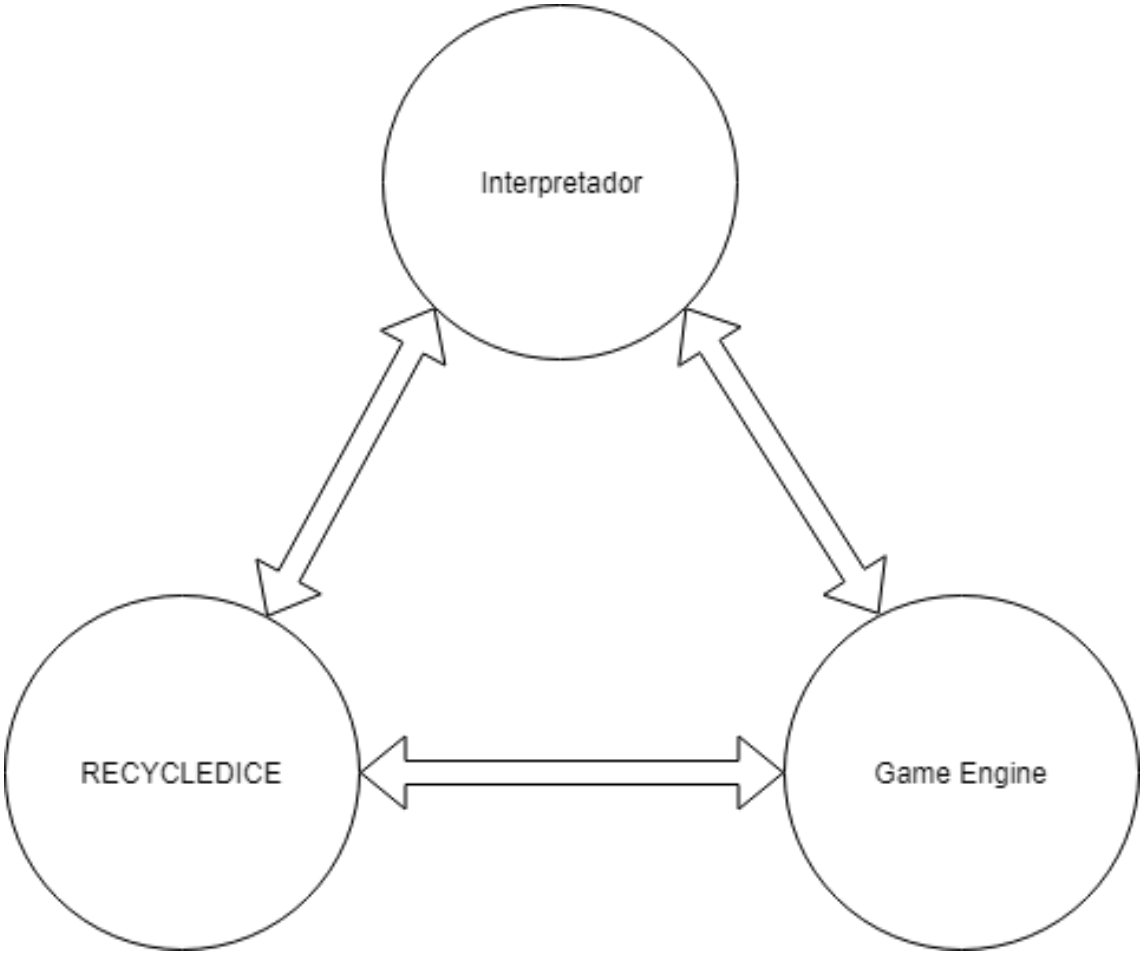

Figura 3.2 – Dependência entre os componentes da arquitetura

Um fator importante da nova linguagem é que ela corresponde a uma extensão da linguagem original, um adicional, ou seja, a alteração da gramática não impactou na necessidade de alteração de jogos anteriormente descritos.

3.2.1 RECYCLEDICE

A extensão da gramática, que chamamos de RECYCLEDICE, permitiu descrever jogos contendo o elemento de dados. Para representar e simular o ato de jogar, o estado do jogo teve que ser modelado e atualizado de acordo, fazendo assim com que fosse necessário três alterações principais na *Game Engine*:

1. Criação de um dado n facetado.
2. Lançamento do dado criado.
3. Recuperação do valor do lançamento.

3.2.1.1 Criação de um dado

A primeira alteração tratou da necessidade de criação e instanciação dos dados. A criação e modelagem dos dados precisa responder algumas perguntas:

- Quantos dados vão existir no jogo?
- Qual a quantidade de lados de cada um deles?
- Os dados serão justos ou viciados?

Buscou-se assim uma modelagem que garantisse a criação de múltiplos dados de quantidade de lados diferentes. Ademais, poucos são os jogos que consideram dados viciados como elemento primordial do jogo. Desse modo, na modelagem, foi considerado somente dados justos.

3.2.1.2 Lançamento de um dado

A segunda alteração correspondeu ao lançamento do dado, ação correspondente a simular o evento físico do lançamento no tabuleiro, retornando o valor da face voltada para cima no dado[1].

Por considerar somente dados numéricos n facetados no sistema, o valor retornado pelo lançamento do dado é sempre um valor aleatório entre um e n.

[1]Determinados dados possuem um padrão diferente para avaliar o resultado. O **d4**, representado por um tetraedro, apresenta o resultado de acordo com face oculta.

3.2.1.3 Recuperação do valor do lançamento

A última alteração indica que esse valor retornado precisou ser salvo na modelagem do dado. Segundo a seção 2.9, um dado pode servir tanto como pontuação imediata para mediar conflitos quanto como um marcador de pontos.

Desta forma, para nossa modelagem de jogo, após o lançamento de um dado, deve-se conseguir verificar o valor do último lançamento desse dado sempre que desejado.

3.2.2 Dice Storage

Uma vez modelado um dado, decidiu-se por aumentar as possibilidades, planejando o lançamento concomitante de dados de diferentes tamanhos em diferentes locais de armazenamento de dados. Dessa forma, dados foram criados e agrupados em áreas específicas, denominadas *Dice Storage*. Para poder referenciar cada *Dice Storage* criado, decidiu-se na modelagem que cada um deveria possuir uma palavra chave como identificação, garantindo assim que novos lançamentos com o *Dice Storage* especificado fossem realizados e que o valor do lançamento fosse recuperado.

Essa última extensão permitiu que duas funcionalidades fossem adicionadas. A primeira consistiu em garantir que combinações diferentes fossem alcançadas, permitindo modelagem de somatório de dados com quantidade de lados diferentes. A segunda correspondeu às diferentes localizações de dados, onde o designer do jogo ou o próprio jogador poderia escolher diferentes combinações de dados para determinadas situações. Levando para o conceito de RPG de mesa, como um exemplo dessas duas funcionalidades, um mestre determina que para atacar deve ser utilizado o conjunto de dados "ATAQUE", sendo necessário rolagem de dados **2d6 + d5**. Em contrapartida, para defesa, o conjunto de dados "DEFESA" precisa ser escolhido, sendo aplicada rolagem de dados **3d10**. Com essa extensão, aumentou-se a quantidade de jogos que podem ser modelados.

A imagem 3.3 apresenta a modelagem das classes em um diagrama UML. A representação da classe *Game* foi simplificada para melhor entendimento da estrutura. As classes *Die* e *Dice Storage* correspondem às classes adicionadas neste projeto.

3.3 Estatísticas de turno

Com o intuito de poder avaliar automaticamente um jogo, é necessário realizar uma análise estatística dos registros gerados pelo sistema. Como proposto por (ALTHÖFER, 2003), percebeu-se a importância de salvar as informações relevantes das simulações, para serem enviadas ao avaliador. Portanto, considerou-se interessante

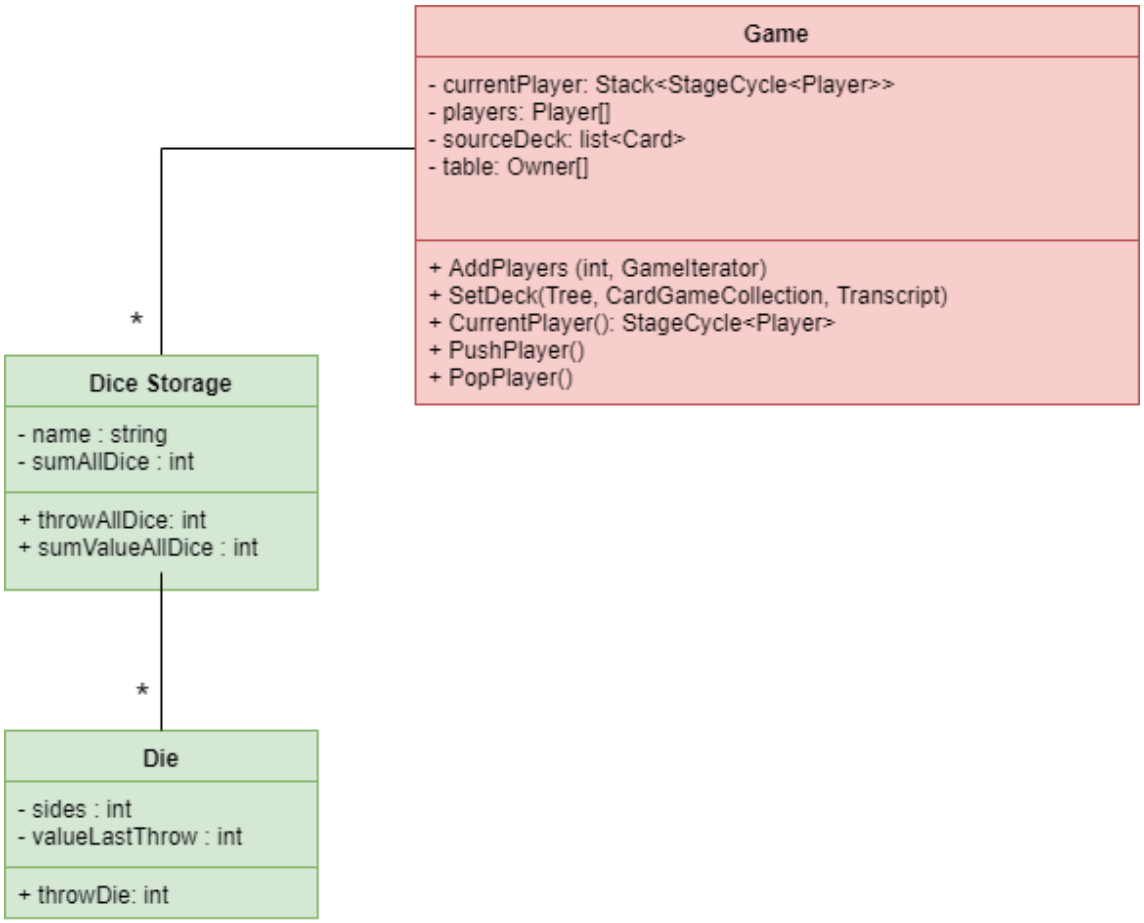

Figura 3.3 – Estrutura das classes adicionadas no RECYCLEDICE

que a ferramenta permitisse recuperar estatísticas de rodadas e turno. Optou-se por extrair, a cada rodada, a pontuação atual dos jogadores.

Em relação ao sistema utilizado como base, é necessário tecer algumas críticas para entender a motivação deste subcapítulo em específico. A gramática do RECYCLE não possui uma definição explícita do que corresponde ao turno de jogo. Para iterar sobre os jogadores, a gramática RECYCLE utiliza a regra *stage*. Essa regra só pode ser derivada da regra *game* ou da própria regra *stage*. Assim sendo, para poder declarar uma regra *stage* em um jogo e, por consequência, iterar sobre os diferentes jogadores, é necessário declarar um *stage* mais externo. Considere agora que a regra *stage* define uma rodada. A partir da regra *stage*, é possível definir uma rodada que itera sobre cada um dos jogadores. Como um *stage* pode ser derivado de outro *stage*, conclui-se que uma rodada pode possuir outras rodadas dentro dela, o que tecnicamente faria o *stage* mais externo não corresponder sempre a uma rodada. Dessa linha de raciocínio, conclui-se que na linguagem existe uma carência em relação a definição do conceito de turno.

Visto que o sistema não possui uma documentação bem estruturada e como consequência dessa não elucidação de qual regra corresponde à rodada, indicar o

início ou término de cada rodada, para fins estatísticos, corresponde a uma tarefa não trivial. A partir desta linha de raciocínio, essa indicação precisou ser realizada na gramática e na descrição.

Isto posto, esta dissertação apresenta uma nova extensão na linguagem que permitiu que as estatísticas fossem salvas com informações de turno. Uma regra nova foi desenvolvida na gramática, a *saveturnstats*, responsável por indicar o momento exato do jogo em que o designer deseja salvar as informações de pontuação de cada jogador da rodada. Para fazer uso dessa regra, dois marcadores precisam ser passados como parâmetros: um marcador de pontuação do jogador e uma marcador de rodada atual. O designer deve ser capaz de atualizar esses marcadores de acordo com a informação de jogo que deseja extrair.

A figura 3.4 mostra um exemplo do tabuleiro de jogo com as mecânicas desenvolvidas no RECYCLEDICE. O *Dice Storage* constituindo um elemento de armazenamentos de dados além da indicação dos dois marcadores necessários, de pontuação SCORE e de rodada ROUND.

3.4 Modelo de jogo para métricas

Para realizar a avaliação qualitativa de um jogo, escolheu-se a ferramenta de avaliação disponibilizada por MANGELI (2016). Como explicado na seção 2.10, esse trabalho disponibilizou um *framework* de avaliação a partir de métricas de estética baseadas em turno. Por sua entrada considerar jogos em turno, foi necessário garantir que as simulações gerassem estatísticas compatíveis com a estrutura definida pelo *framework*, processo detalhado na seção anterior.

De modo igual, além deste processamento, foi adicionado um novo modelo das classes de jogo à ferramenta de avaliação, responsável por receber os dados no formato exigido. Desta forma, as classes de *Match* e *DiceGame* foram geradas a partir do modelo de jogo genérico existente na ferramenta. Não foi mais necessária a criação de um modelo específico a cada jogo para servir de entrada para a ferramenta, bastando somente usar o novo modelo de jogo como intermediário para cada descrição de jogo interpretada e simulada. A imagem 3.5 apresenta o diagrama de classes UML com o modelo de implementação das classes. As classes implementadas neste trabalho correspondem às classes em verde. O método setGameStruck da classe *DiceGame*, que representa o jogo, fica responsável por montar a estrutura necessária para a aplicação das métricas: salvando vencedor, pontuação por rodada, número de rodadas, última rodada, além da informação de pontuação de cada rodada em tuplas. A classe *Match*, que representa uma partida, recupera a estrutura de arquivos original e carrega as informações em tuplas de jogador e pontuação, através da *ItemDiceTuple*, para cada round.

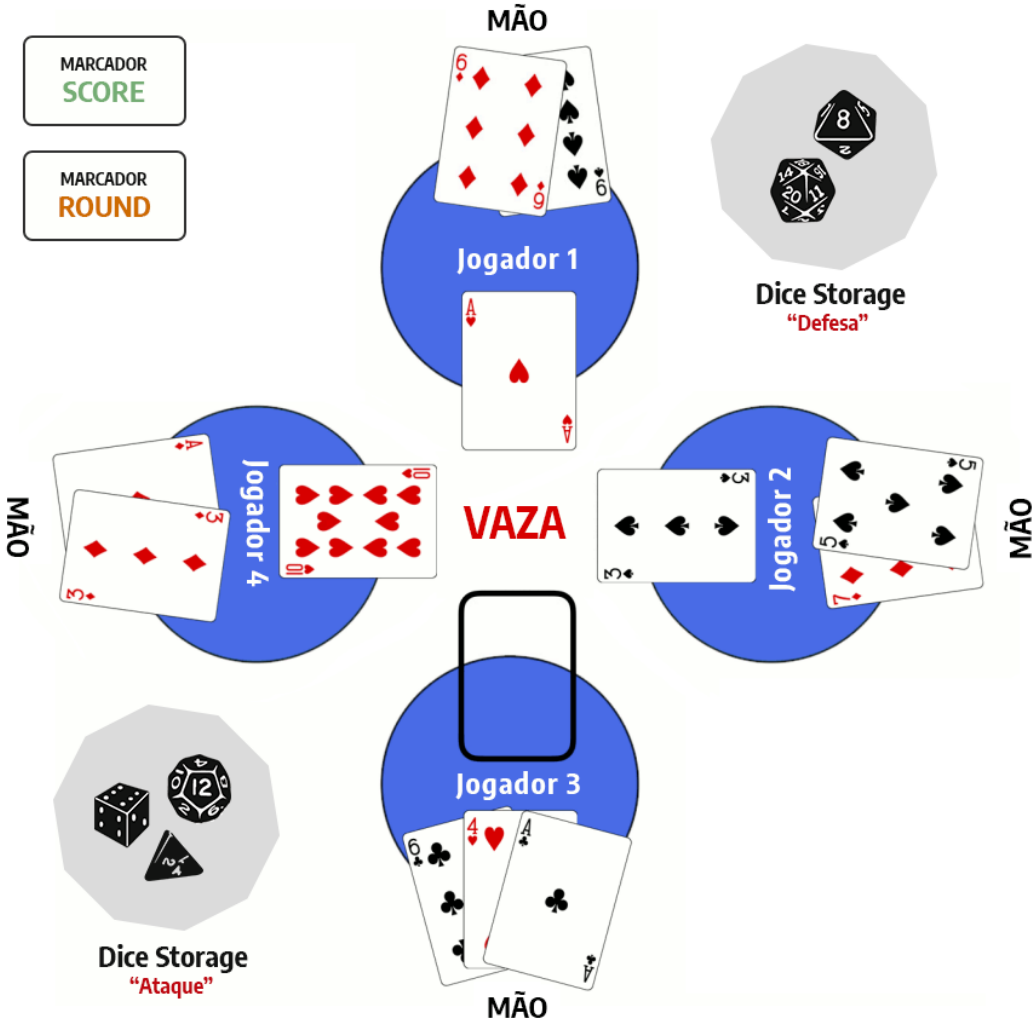

Figura 3.4 – Exemplo de Tabuleiro do RECYCLEDICE, baseado e adaptado de (BELL & GOADRICH, 2016)

Assim, a partir deste modelo, foi possível aplicar as métricas de avaliação de estética correspondentes a Drama, Mudança de líder e Incerteza. Primeiramente, foi avaliada uma partida em específico, verificando o comportamento dos jogadores durante a partida, analisando os gráficos de posição e de pontos. Posteriormente, simulações representando várias partidas foram apresentadas, com o intuito de entender e avaliar o comportamento das métricas diante uma grande quantidade de dados. Desta forma, realizando a avaliação da estética do jogo sobre determinados critérios.

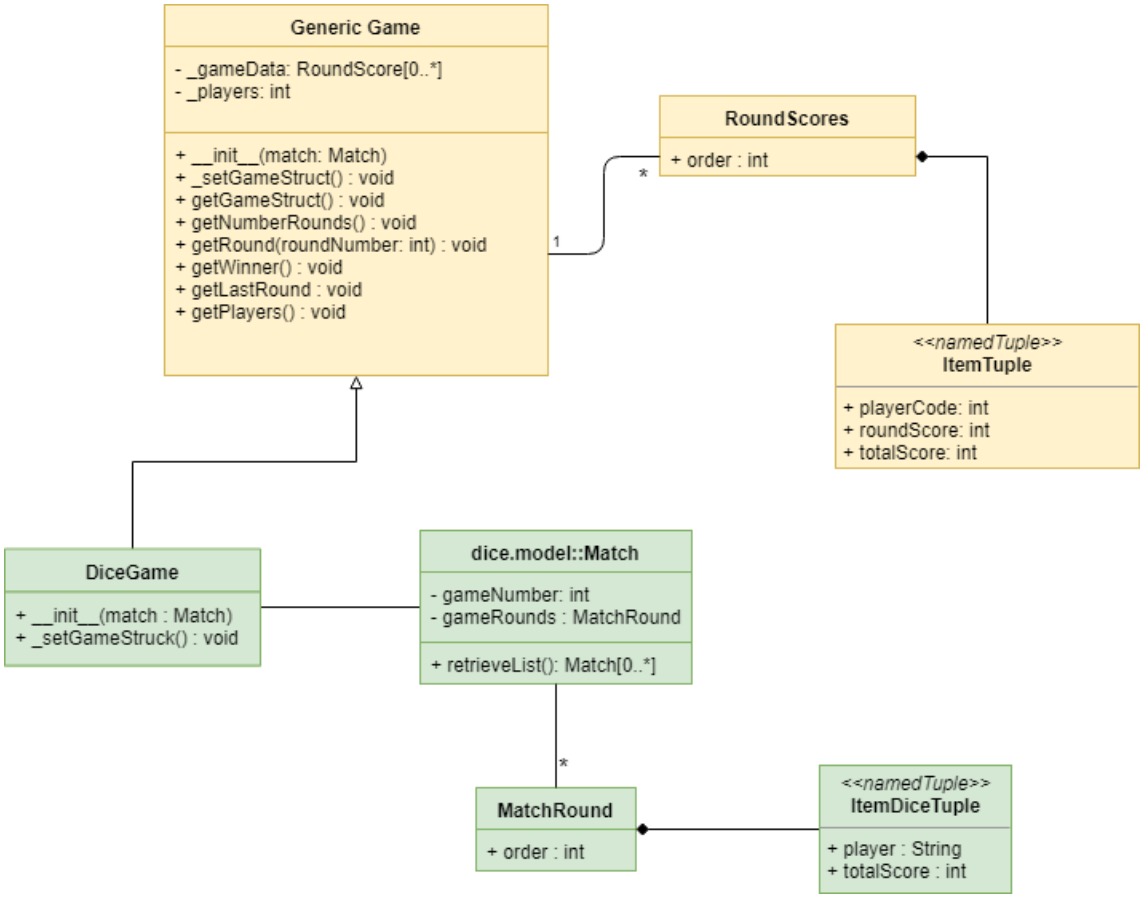

Figura 3.5 – Modelo de implementação das classes adicionais, baseado em (MANGELI, 2016)

Capítulo 4

Implementação

Esse capítulo apresenta a implementação das ideias propostas no último capítulo. Assim, aqui é mostrada a implementação da extensão da linguagem RECYCLE a partir da modelagem do elemento de dados. Esse capítulo também é responsável por expor a adaptação do Cardstock que permitiu o salvamento de estatísticas por turno. Por fim, ele revela a implementação realizada para que o *framework* de avaliação recebesse os novos jogos.

A linguagem modificada RECYCLEDICE foi aplicada a uma descrição de jogos, modelando assim um jogo com dados. Em seguida, simulações foram executadas e estatísticas sobre o jogo geradas, a partir do novo simulador GameStock. Posteriormente, a nova implementação da ferramenta de avaliação permitiu a avaliação de uma grande quantidade de estatísticas de jogos genéricos.

4.1 Implementando uma nova linguagem

O RECYCLEDICE foi desenvolvido com o intuito de implementar a modelagem proposta, uma extensão da linguagem RECYCLE que permite a modelagem da classe de jogos de dados, assim como permite descrever jogos de cartas com dados. Essa modificação foi realizada em três frentes:

- Na adição e adaptação de regras no arquivo da gramática, Recycle.g4.
- Na implementação dos novos métodos do interpretador responsáveis por fazer a travessia da árvore nas novas regras, no sistema *CardStock*.
- Por fim, na atualização da modelagem e simulação do jogo, na *CardEngine* do *CardStock*, que foram denominados de *GameEngine* e *GameStock* respectivamente.

A linguagem RECYCLE foi definida na gramática **Recycle.g4** e é a partir dessa gramática que a linguagem RECYCLEDICE foi desenvolvida. A ferramenta ANTLR processa a gramática para gerar um analisador sintático.

O simulador *GameStock* foi dividido em dois componentes essenciais: o primeiro corresponde ao *GameEngine*, que simula o tabuleiro de jogo com cartas, jogadores, espaço para armazenamento das fichas, além de realizar as ações do jogo como embaralhar baralho, mover cartas, dentre outros.

O segundo componente é o interpretador, que permite visitar os nós da árvore de análise sintática gerada de acordo com a regra lida na descrição, ou seja, é o interpretador que processa a descrição do jogo e atualiza o *GameEngine* de acordo com a implementação de cada regra.

4.1.1 Criação do componente de Dado

Começando pelo *GameEngine*, para a adição do componente de dado, foi necessária a criação da classe *Die* no sistema, responsável pela modelagem do dado. A classe possui os atributos quantidade de lados e valor do último lançamento, assim como um método de lançamento do dado, que retorna um valor entre um e a quantidade de faces do dado, além de salvar o valor do último lançamento. Além disso, foi criada a classe *Dice Storage*, contendo uma palavra chave identificadora, uma lista de dados e um valor indicando a soma do último lançamento. Dois métodos são importantes para essa classe, o *ThrowAllDice* que aplica o lançamento de todos os dados daquele local e o *SumValueAllDice* que retorna o valor do somatório do último lançamento de dados. Por fim, um dicionário de *DiceStorage* foi instanciado na classe *Game*, mantendo uma *string* como chave de cada *DiceStorage* e permitindo que uma quantidade arbitrária de locais de armazenamento de dados seja criada.

Para explicar as alterações no interpretador, é fundamental entender primeiro as mudanças realizadas na gramática. Assim, explicamos previamente as alterações na gramática da linguagem RECYCLE para gerar o RECYCLEDICE. A primeira alteração consistiu na criação da regra *diecreate*, responsável por criar locais de armazenamento dos dados. A regra *diecreate* corresponde a uma alternativa de produção na regra *setup* às regras de *deckcreate* e *repeat*. A imagem 4.1 apresenta o diagrama de sintaxe que permite o uso da regra de criação de um dado.

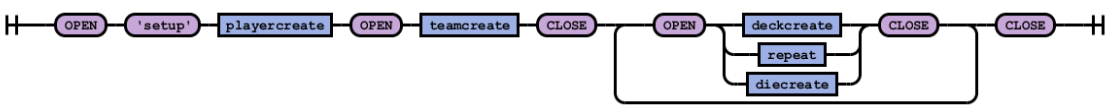

Figura 4.1 – Diagrama de sintaxe da regra *setup*

A regra *diecreate* possui dois *tokens* que permitem identificar a regra, que são *'create'* e *'diestorage'*. Após esses símbolos, um símbolo não terminal de nome *var* representa a palavra chave identificadora do local. Por fim, o símbolo não terminal *die* pode ser repetido uma ou mais vezes, criando cada dado. O diagrama de sintaxe da regra *diecreate* é apresentado na imagem 4.2.

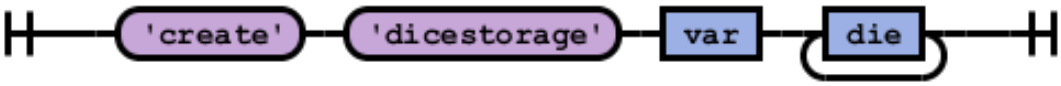

Figura 4.2 – Diagrama de sintaxe da regra *diecreate*

Consequentemente, a regra de criação de cada dado *die* é composta por um *token* de abertura, um *token* de fechamento e entre eles um não-terminal *int* indicador de valor inteiro, correspondente à quantidade de faces do dado. A figura 4.3 apresenta o diagrama de sintaxe da regra *die*.

Figura 4.3 – Diagrama de sintaxe da regra *die*

Posteriormente, foi necessária a atualização do interpretador, responsável por visitar os nós da árvore de análise sintática das novas regras geradas. Ao derivar a regra de *setup*, verifica-se a existência da regra *diecreate*. Para cada *diecreate* descrito, o método *ProcessDiceStorage* é chamado, responsável pela criação de cada local de armazenamento do dado.

Esse método é responsável pelas seguintes funções:

1. Recuperar o texto que serve como chave identificadora do local, identificado pelo símbolo *var*.

2. Criar uma nova instância de local de armazenamento dos dados, o *Dice Storage*.

3. Para cada dado especificado, criar uma instância da classe *Die* com a quantidade de faces correta.

4. Adicionar cada um dos dados criados no *Dice Storage*.

5. Adicionar o *Dice Storage* no dicionário de locais que armazenam dados, na classe Game.

A figura 4.4 apresenta um ramo de exemplo de árvore de análise sintática gerada a partir de uma descrição. Através das derivações da regra *setup*, um *Dice Storage* de nome GAMEDICE é criado, contendo um dado **d6**.

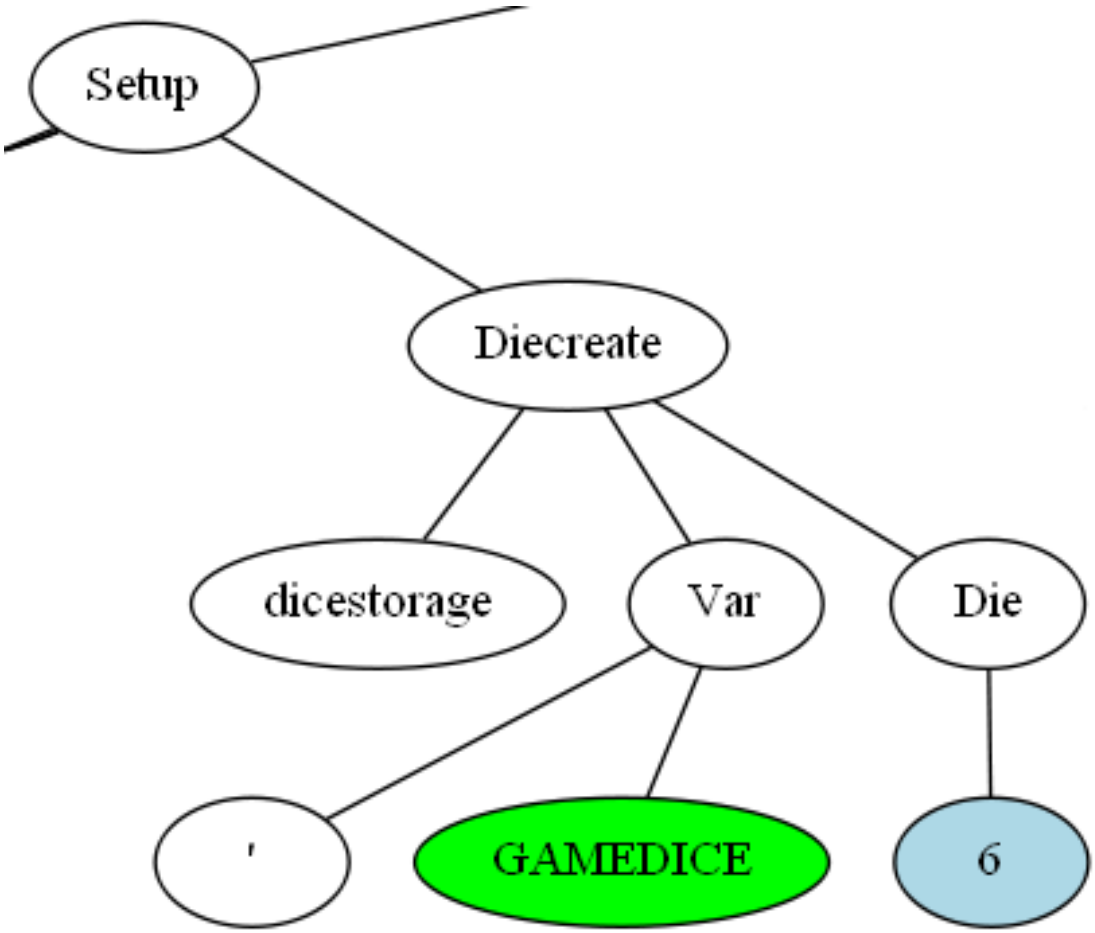

Figura 4.4 – Exemplo da árvore de análise sintática da regra de criação de um *Dice Storage*

4.1.2 Lançamento de Dados

Essa seção é responsável por explicar como foi implementado o lançamento dos dados. A seção anterior apresentou a modelagem realizada para o *GameEngine*, restando mostrar as alterações na gramática e no interpretador.

Em relação à gramática, a adição de uma alternativa de ação na regra *action* do jogo permitiu que dados fossem lançados. Essa alternativa corresponde à regra *throwalldice*, como mostrado na figura 4.5. Uma *action* representa uma ação realizada pelo computador, ou seja, tecnicamente quem realiza a ação de jogar o dado é o computador e não o jogador.

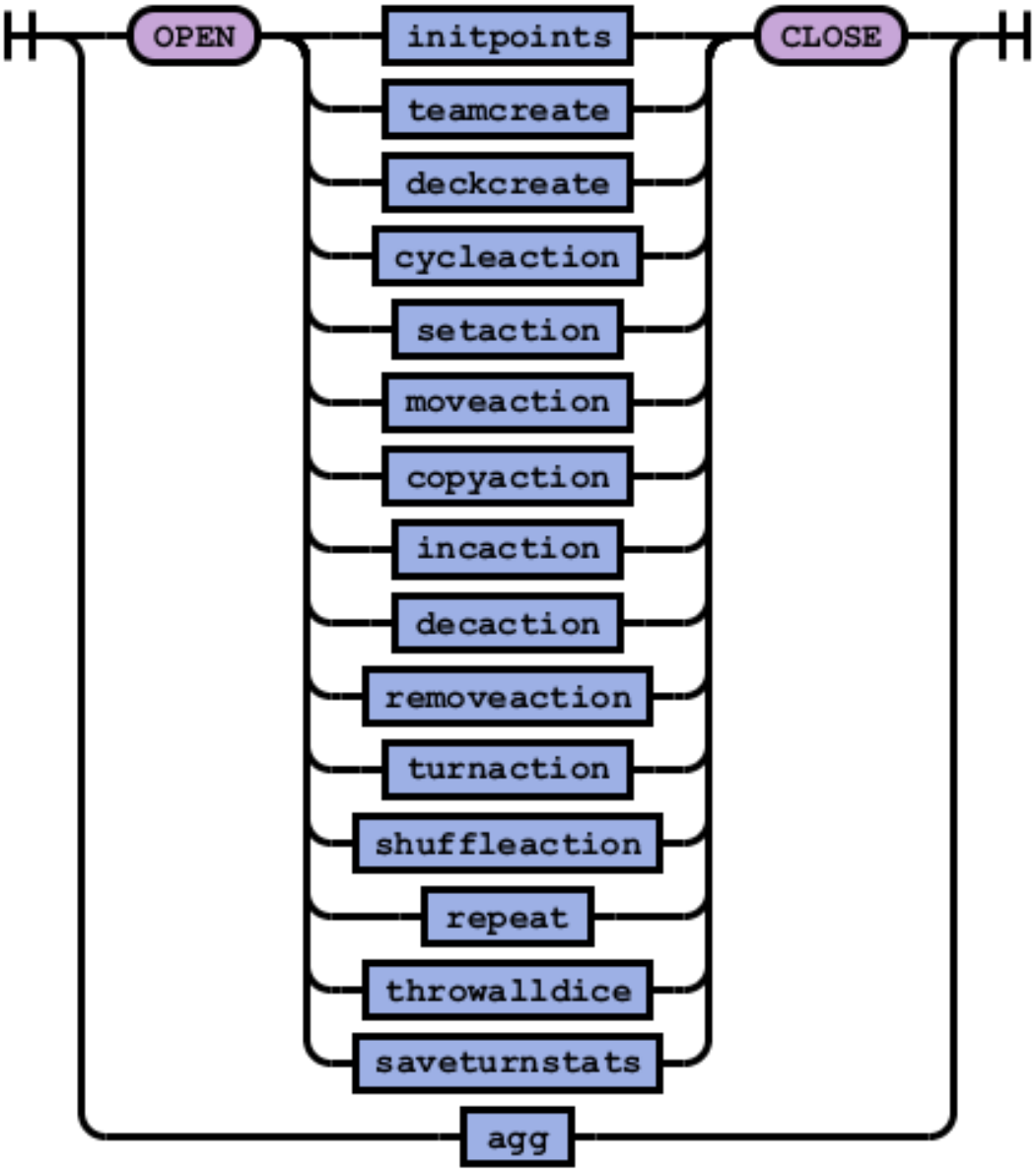

Figura 4.5 – Diagrama de sintaxe da regra *action*

A regra *throwalldice* é composta por um *token 'throwalldice'* e uma variável *var* indicando a chave do local de armazenamento de dados cujo lançamento deve ser realizado. A imagem 4.6 apresenta o diagrama de sintaxe da regra.

Figura 4.6 – Diagrama de sintaxe da regra *throwalldice*

Quanto ao iterador de árvore, ao processar a regra *throwalldice*, o *Dice Storage* indicado é encontrado, realizando a chamada ao método *ThrowAllDice* correspondente, fazendo assim o lançamento de todos os dados do local e salvando a face retornada por cada um dos dados, assim como o somatório dos valores das faces.

4.1.3 Recuperando valor do lançamento

Para recuperar o valor do último lançamento realizado, a gramática foi modificada, adicionando a regra *dievalue*, uma alternativa à regra de *int*. Assim, em qualquer regra que permite valores inteiros, é possível substituir esse valor inteiro pela nova regra adicionada.

A figura 4.7 apresenta a regra *dievalue*. A regra *dievalue* possui uma estrutura de *token* de abertura *OPEN*, seguido por um *token 'dievalue'*, tendo diretamente em seguida uma variável *var* indicadora do nome do *Dice Storage* e finalizando com um *token* de fechamento *CLOSE*. Assim, qualquer inteiro pode ser substituído pelo valor de lançamento do conjunto de dados.

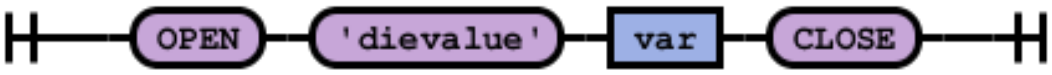

Figura 4.7 – Diagrama de sintaxe da regra *dievalue*

Em relação à implementação no interpretador, ao recuperar o valor de um inteiro, verifica-se se a regra do inteiro corresponde à regra de *dievalue*. Caso positivo, é realizada a busca pelo *Dice Storage* de nome correspondente, chamando o método *SumValueAllDice* que retorna o valor do último lançamento.

4.2 Extraindo estatísticas

Para extrair as estatísticas de turno do simulador, criou-se uma regra na gramática capaz de informar o momento exato em que o valor da pontuação dos jogadores deve ser retornado. A regra criada para realizar o salvamento foi a *saveturnstats*. Assim como a regra *throwalldice*, esta regra também corresponde a uma possível derivação da regra *action*, correspondendo assim a uma ação do computador, como mostrado na figura 4.5. Desta forma, a regra *saveturnstats* foi concebida, demandando duas variáveis de marcadores representados pelas regras *namegr*. A primeira regra de *namegr* deve corresponder à variável que armazena a pontuação atual dos jogadores e a segunda *namegr* faz referência à variável que armazena o número da rodada atual do jogo. A imagem 4.8 apresenta o diagrama de sintaxe da regra *saveturnstats*. O

token 'roudnumber' foi criado para indicar que a segunda variável corresponde ao marcador turno atual.

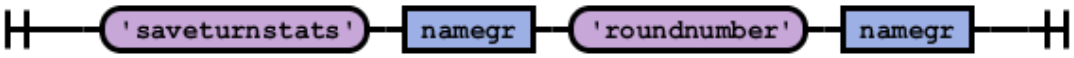

Figura 4.8 – Diagrama de sintaxe da regra *saveturnstats*

Também foi necessário processar essa regra no interpretador, que recebeu as duas variáveis, de pontuação e de rodada, chamando um método da classe *Game*, o *WriteToStatFile*. Este método itera sobre todos os jogadores, recuperando a pontuação atual de cada um deles. A cada iteração, esses valores são adicionados em triplas com: número da rodada atual, nome do jogador atual e pontuação deste jogador. Essas triplas são exportadas e salvas no formato *Comma-Separated value*, seguindo o padrão **rodada;jogador;pontuação**.

4.3 Adaptando Modelo de Jogos para a *Framework* de Avaliação

Com o intuito de fornecer ao *framework* de avaliação as simulações das partidas, adaptações na ferramenta foram necessárias. Para criar o novo modelo de jogo, foi feito uso do modelo de estrutura de jogo genérico da ferramenta. A figura 3.5 apresenta o modelo de jogo criado. A classe *Match* foi criada no pacote *dice.model*, adaptada para operar sobre informação recuperada de um arquivo no formato *Comma-separated values*. Nessa classe foi implementado o método *retrieveList*. Esse método é responsável por:

1. Leitura do arquivo no formato *Comma-separated values*, contendo as informações da simulação de um jogo.

2. O modelo da classe *GenericGame* demanda que os jogadores estejam ordenados na rodada. Isso implica na realização da ordenação por rodada e por pontos, de forma a deixar o jogador com mais pontos na rodada aparecendo como primeiro da rodada.

3. Para cada linha lida do arquivo, geração de tuplas com o identificador do jogador e sua pontuação na rodada.

4. Agrupa as tuplas por *MatchRound*, rodadas da partida, que por sua vez são agrupados em *Match*, partida. Cada um dos jogos simulados consiste assim em uma partida, instância da classe *Match*.

A partir da classe *GenericGame*, especifica-se a classe de jogos *DiceGame*, que lida com o *dataset* dos jogos gerados através da simulação. Recebendo um objeto do tipo *Match*, o método *setGameStruct* monta a estrutura do atributo *gameData*. A estrutura do *gameData* corresponde ao número da rodada e um conjunto de tuplas que armazenam o código do jogador, sua pontuação na rodada e sua pontuação total.

Para calcular a métrica de uma simulação, a instância da classe *DiceGame* foi passada por parâmetro para um método que calcula uma estética específica desejada. Foi realizado o cálculo com as métricas de Drama por Pontos, Drama por Posição, Drama por Caminho, Mudança de Liderança, Incerteza por Entropia e Incerteza por PDD. Para calcular as métricas de várias simulações, múltiplas instâncias da classe *DiceGame* foram geradas, iterando sobre cada uma delas e calculando o valor da métrica para cada partida, gerando assim uma grande quantidade de métricas calculadas. Para avaliar o comportamento da métrica de acordo com o jogo, os dados obtidos foram organizados graficamente em forma de histograma. Desta forma, foi possível observar a distribuição de cada uma das métricas de acordo com as simulações.

4.4 Descrição de Jogos de Teste

Para verificar o funcionamento da estrutura concebida e desta forma realizar uma prova de conceito, foi necessário a descrição de jogos de teste.

Dois jogos foram concebidos e descritos. Eles foram denominados: DicePoints e ThrowDiceMatchCard.

4.4.1 DicePoints

DicePoints é um jogo que possui somente o componente de dados. Quatro jogadores competem em um jogo de lançamento de dado, cada um lançando um conjunto de dados a cada rodada e recuperando esse valor. O valor dos dados é somado à pontuação do jogador. O jogador com mais pontos ao final de um determinado número de rodadas ganha. Para ilustrar os elementos da nova linguagem, a descrição do DicePoints é apresentada no código 4.1, especificando a descrição completa de um jogo em RECYCLEDICE.

Entre as linhas 2 e 6, a quantidade de jogadores foi definida, os times foram definidos como jogadores individuais e na linha 5 o *Dice Storage* de nome GAMEDICE foi instanciado, com a criação de dois dados *d5*.

```
(game
 (setup
  (create players 4)
  (create teams (0) (1) (2) (3))
  (create dicestorage 'GAMEDICE (5)(5) )
 )
 (do
  (
   (set (game sto ROUNDS) 0)
   (set (game sto MOVE) 1)
  )
 )
 (stage player
  (end (== (game sto ROUNDS) 50))
   (stage player
    (end (== (game sto MOVE) 5))
     (do
      (
       (throwalldice 'GAMEDICE)
       (inc ((current player) sto SCORE) (dievalue 'GAMEDICE))
       (inc (game sto MOVE) 1)
      )
     )
   )
   (do
    (
     (set (game sto MOVE) 1)
     (inc (game sto ROUNDS) 1)
     (saveturnstats SCORE roundnumber ROUNDS)
     (cycle current previous)
    )
   )
 )
 (scoring max ((current player) sto SCORE))
)
```

Código 4.1 – Descrição do jogo DicePoints em RECYCLEDICE

Da linha 7 até a linha 12, os marcadores de número de rodada, **ROUNDS** e de quantidade de movimentos na rodada, **MOVE**, foram iniciados com valores 0 e 1 respectivamente.

Na linha 13, um estágio iterando sobre os jogadores foi iniciado. A condição de parada aparece na linha seguinte, terminando o estágio caso o marcador de rodada alcance o valor de 50. Desta forma, o jogo finaliza após 50 rodadas.

Logo a seguir, outro estágio foi aninhado na linha 15, com a condição de parada de terminar quando o marcador de rodada alcançar o valor 5. Ou seja, após iterar sobre cada um dos 4 jogadores.

A partir da linha 17, para cada jogador, três ações foram realizadas: o lançamento do conjunto de dados na linha 19, o valor do lançamento dos dados foi recuperado através da regra *dievalue* e foi por sua vez somado à pontuação do jogador atual, **SCORE** , na linha 20, e o marcador de movimento foi acrescentado de um na linha seguinte.

Após iterar sobre todos os jogadores, a partir da linha 25 até a linha 32, uma nova sequência de ações foi apresentada. O marcador de movimentos é reiniciado, o número da rodada é incrementado, as estatísticas da rodada são salvas na linha 29, com base nos marcadores **SCORE** e **ROUNDS**, e a ordem de turno é reiniciada na linha 30.

Por fim, o vencedor é decidido na linha 34, que corresponde ao jogador com maior **SCORE**.

4.4.2 ThrowDiceMatchCard

ThrowDiceMatchCard é um jogo que mescla os componentes de dados e cartas. Quatro jogadores competem em um jogo de vaza com lançamento de dados. O baralho consiste em 25 cartas, com cinco cartas de cada valor, numeradas de 1 a 5. O dado utilizado é um dado de 5 faces. 5 cartas são entregues para cada um dos jogadores e o jogo se dá em 5 rodadas. A cada rodada, um dado é lançado e os jogadores devem escolher uma carta para jogar. Se o valor da carta for igual ao valor do dado, o jogador soma um ponto. Caso contrário, não soma ponto. O jogador que tiver mais pontos ao final de todas as rodadas, ganha. O código desse jogo pode ser lido no apêndice A.2.

Foi utilizado o número 5 para exemplificar quantidade de rodadas, valor do dado, quantidade de cartas na mão e valores das cartas. Contudo, por ser um jogo conceitual, esse valor pode e deve ser alterado pelo designer de jogos, buscando o valor que gere as melhor experiência.

Capítulo 5

Resultados

Este capítulo apresenta os experimentos realizados e os resultados obtidos a partir deles. Foi avaliado o funcionamento da ferramenta proposta em 3 e implementada em 4, primeiramente gerando as simulações e sua estatística e posteriormente verificando os valores de estéticas em relação aos valores obtidos nas simulações.

Para os experimentos, dois jogos foram utilizados, DicePoints e ThrowDiceMatchCard, descritos na sessão 4.4. Cada um dos jogos foi descrito em RECYCLEDICE e serviu como entrada para a GAME ENGINE, responsável pelas simulações. Foram executadas 10 mil simulações por jogo, para cada variante desse jogo. Cada simulação gerou um arquivo de estatística, que serviu como entrada para a *framework* de avaliação de Mangeli. As métricas foram calculadas para o conjunto de simulações de um jogo, e sua distribuição foi analisada.

5.1 DicePoints

O primeiro jogo analisado foi o DicePoints. Sete variantes do jogo foram criadas e comparadas. Variamos a quantidade de rodadas, a quantidade de faces dos dados e a quantidade de dados. Para simplificar a notação, chamamos cada variante de kR + notação padrão de dado, onde k é a quantidade de rodadas na partida. Desta forma, para uma partida com 10 rodadas usando um dado de 6 faces, a notação fica **10Rd6**. A tabela 5.1 apresenta as diferentes variantes do jogo. Cada uma das variantes serviu de entrada para o simulador, e as estatísticas de turno das 10 mil simulações foram geradas.

5.1.1 Porcentagem de Vitórias

Foi realizada uma comparação da quantidade de vitórias entre cada jogador com o objetivo de garantir o bom funcionamento da ferramenta e do jogo. De acordo com o gráfico em setores 5.1, podemos verificar que a ferramenta e o jogo não são

Variante	Quantidade de Rodadas	Conjunto de dados utilizado
10Rd6	10	d6
10Rd10	10	d10
10R2d5	10	2d5
50Rd6	50	d6
50Rd10	50	d10
50R2d5	50	2d5
50Rd50	50	d50

Tabela 5.1 – Variantes do DicePoints simuladas

enviesados em relação ao vencedor, visto que todos os jogadores possuem cerca de 25% de vitórias. Isso se dá, como esperado, pela simetria do jogo, dado que todos os jogadores possuem as mesmas regras, e pela aleatoriedade dos dados, garantindo que todos possuem a mesma probabilidade de vitória. No caso de empates na simulação, a ferramenta de avaliação seleciona o primeiro dentre eles como vencedor. Desta forma, é interessante garantir que o jogo seja pouco propenso a empates. A variante **10Rd6** do jogo apresenta esse problema, como mostrado no gráfico em setores 5.2. Por causa da baixa quantidade de rodadas e pela quantidade de faces ser pequena, empates são muito comuns, fazendo com que o *Player 1* possua muito mais vitórias do que os outros jogadores.

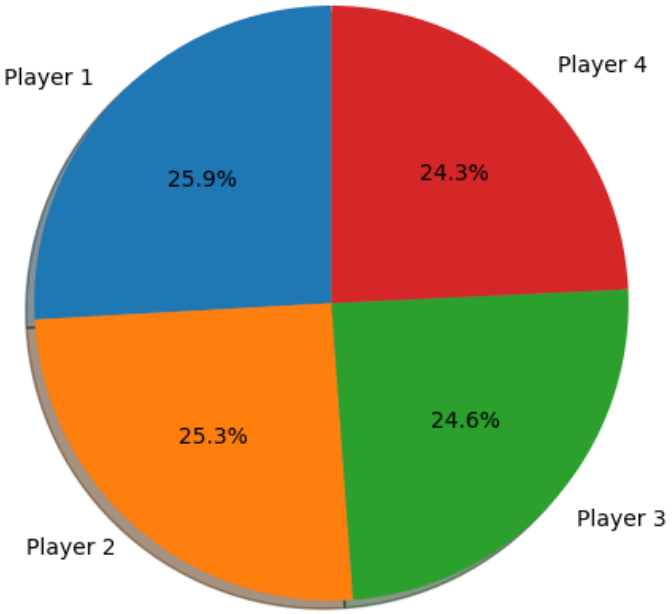

Figura 5.1 – Partidas vencidas na variante **10Rd10**

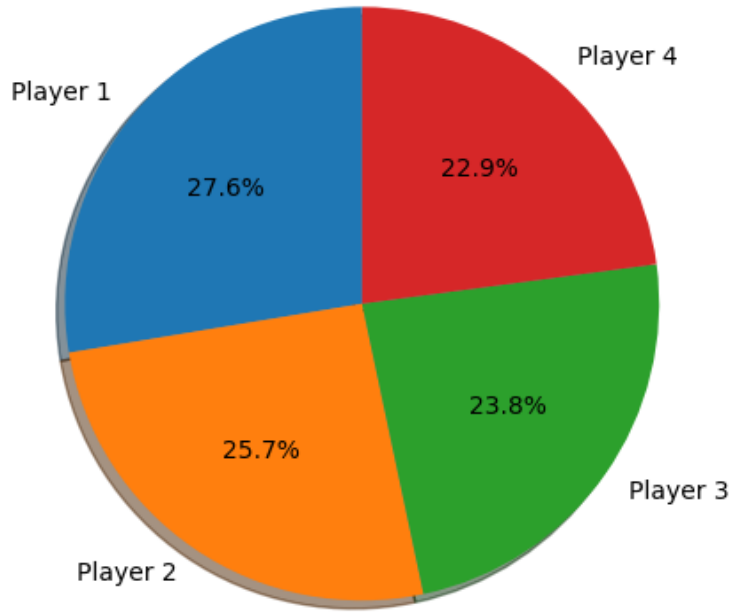

Figura 5.2 – Partidas vencidas na variante **10Rd6**

5.1.2 Avaliação de uma partida

Para aplicar as métricas de estética, foi realizada a importação das estatísticas através do novo jogo modelado na *framework* de avaliação. Assim, para verificar o funcionamento da ferramenta, uma partida em isolado foi avaliada. O gráfico 5.3 apresenta a variação de pontuação dos jogadores em uma partida na variante **10Rd6**. Percebe-se que a pontuação do jogador vencedor ficou distante do líder durante algumas rodadas. Por sua vez, o gráfico 5.4 apresenta a variação de posição dos jogadores durante a mesma partida, na qual é possível visualizar que o vencedor ficou muitas rodadas na última posição e que ele chegou à liderança somente na 9º rodada.

A tabela 5.2 apresenta os valores de Drama da partida apresentada anteriormente. Como o Drama varia de 0 a 1, estima-se que estes valores sejam altos. De fato esses valores são altos, dado que na próxima seção os resultados sobre a distribuição do Drama serão apresentados e verificamos que todos os valores de Drama para essa partida se encontram acima da média.

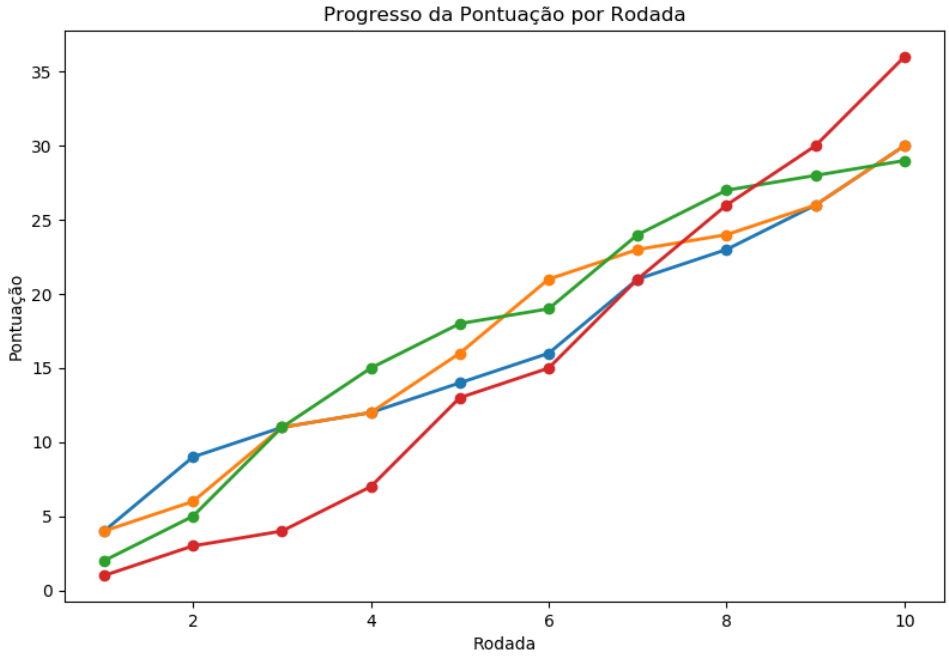

Figura 5.3 – Partida de exemplo de progresso de Pontuação por Rodada, **10Rd6**

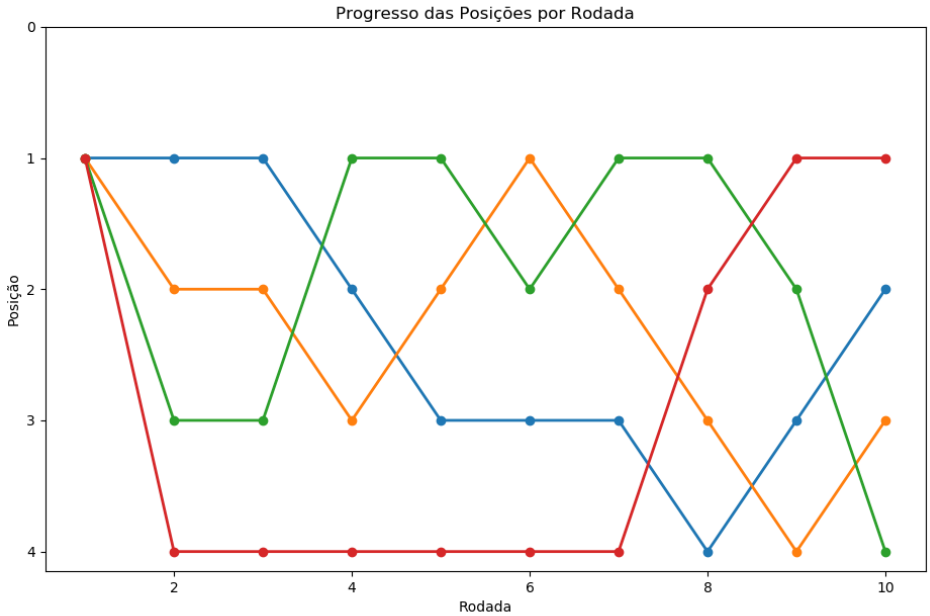

Figura 5.4 – Partida de exemplo de progresso da Posição por Rodada, **10Rd6**

5.1.3 Distribuição do Drama

Para cada variante do DicePoints, calculou-se os valores de estética de Drama por Pontos, Drama por Posição e Drama por Caminho das 10 mil partidas simuladas.

Drama por Pontos	0.6022
Drama por Caminho	0.6913
Drama por Posição	0.8202

Tabela 5.2 – Valores de Drama para uma partida de exemplo do DicePoints, **10Rd6**

Em algumas situações, o Drama é zero. Isso ocorre quando o vencedor da partida foi sempre o líder, não existindo alterações na liderança no decorrer da partida. A tabela 5.3 mostra a quantidade de partidas em que o Drama foi maior que zero e a porcentagem de jogos em que não ocorreu mudança de líder.

Pela tabela, foi possível deduzir que quanto maior o número de rodadas, mais o Drama se mostrou presente. Ou seja, quanto mais longa a partida, mais incomum do vencedor liderar durante toda a partida. Também foi observado que aumentar a quantidade de faces diminui a porcentagem de jogos sem Drama. Por fim, combinar dois dados de cinco faces ao invés de usar um dado de dez faces gerou mais partidas de Drama zero.

	Drama por Pontos		Drama por Posição		Drama por Caminho	
	Quantidade de Simulações com Drama	Porcentagem de Simulações sem Drama	Quantidade de Simulações com Drama	Porcentagem de Simulações sem Drama	Quantidade de Simulações com Drama	Porcentagem de Simulações sem Drama
10Rd6	7945	**20,55%**	8561	**14,39%**	8561	**14,39%**
10Rd10	8207	**17,93%**	8578	**14,22%**	8578	**14,22%**
10R2d5	8116	**18,84%**	8567	**14,33%**	8567	**14,33%**
50Rd6	9504	**4,96%**	9657	**3,43%**	9657	**3,43%**
50Rd10	9545	**4,55%**	9642	**3,58%**	9642	**3,58%**
50R2d5	9496	**5,04%**	9625	**3,75%**	9625	**3,75%**
50Rd50	9650	**3,5%**	9670	**3,3%**	9670	**3,3%**

Tabela 5.3 – Presença do Drama nas simulações do DicePoints

Com o objetivo de avaliar o comportamento do Drama quando ele existe, partidas que não tiveram Drama foram desconsideradas na análise de distribuição do Drama. Para cada variante do jogo, foi realizado o cálculo da média e desvio padrão do Drama. A tabela 5.4 apresenta esses valores para cada Drama de acordo com a variante. Observou-se que os jogo que obtiveram maior média de Drama por Pontos, por Posição e por Caminho correspondem respectivamente ao **10Rd10**, **50Rd10** e ao **50Rd10**. A partir disso, é possível concluir que nas simulações observadas, uma quantidade maior de faces implica em um Drama maior.

A figura 5.5 apresenta o gráfico de dispersão em 3d dos valores de Drama para cada variante. A intensidade de cor de cada ponto indica a profundidade no gráfico, cores mais intensas correspondem aos valores mais próximos enquanto uma cor mais fraca significa uma maior profundidade. Deste gráfico, observa-se que o Drama segue uma estrutura diagonal: para média de Drama por Caminho elevados e média de Drama por Posição elevados, tem-se média Drama por pontos baixo. Da mesma forma, ao possuir média de Drama por Pontos elevada, a variante possui Drama por

	Drama por Pontos		Drama por Posição		Drama por Caminho	
	Média	Desvio Padrão	Média	Desvio Padrão	Média	Desvio Padrão
10Rd6	**0,3766**	0,1358	**0,5888**	0,0898	**0,2343**	0,2022
10Rd10	**0,3832**	0,1360	**0,5892**	0,0904	**0,2365**	0,2033
10R2d5	**0,3221**	0,1152	**0,5919**	0,0907	**0,2397**	0,2055
50Rd6	**0,2886**	0,1040	**0,5932**	0,0784	**0,2486**	0,2108
50Rd10	**0,3006**	0,1026	**0,5954**	0,0796	**0,2521**	0,2120
50R2d5	**0,2436**	0,0873	**0,5942**	0,0791	**0,2487**	0,2099
50Rd50	**0,3096**	0,1049	**0,5938**	0,0782	**0,2508**	0,2100

Tabela 5.4 – Média e Desvio Padrão do Drama das simulações do DicePoints

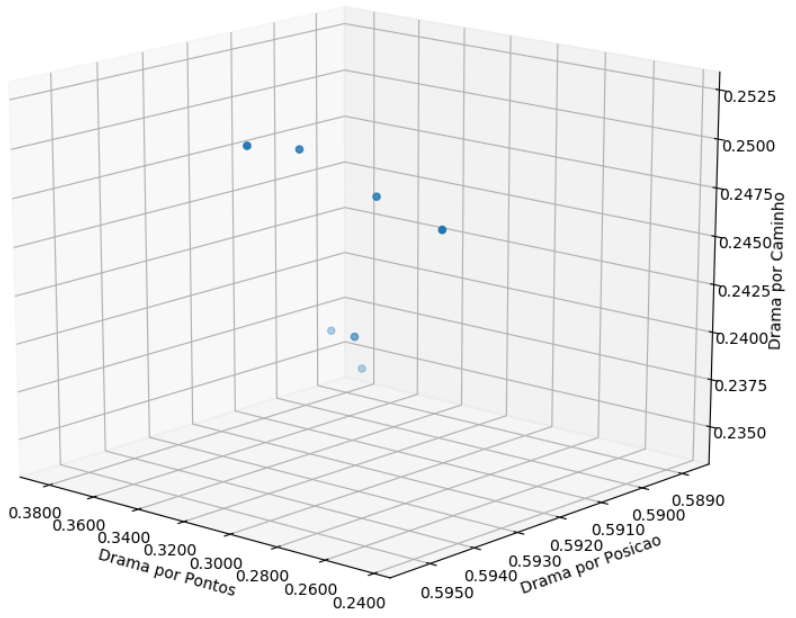

Figura 5.5 – Gráfico de dispersão 3d dos valores de média do Drama para cada variante

Posição e Drama por Caminho baixo. Por conseguinte, observa-se que existe uma relação de correlação entre as diferentes métricas de Drama.

As figuras 5.6, 5.7 e 5.8 apresentam o gráfico da distribuição do Drama por Pontos, do Drama por Posição e do Drama por Caminho, respectivamente, para cada variante. Os gráficos foram gerados a partir do histograma de frequência de cada drama para cada simulação. As curvas foram suavizadas empregando a

estimativa de densidade de kernel, aplicadas no conjunto de dados para estimar a função de probabilidade de densidade. Todas os próximos gráficos desta seção foram suavizados aplicando esta técnica. O eixo das ordenadas indica a frequência de aparecimento do valor da métrica em determinada faixa e varia de acordo com a métrica.

O gráfico da figura 5.6, do Drama por Pontos, apresenta um comportamento que se assemelha ao da distribuição normal. Desta forma, observa-se que as variantes que mais se aproximam dos valores altos de drama correspondem às variantes **10Rd6** e **10Rd10**, com poucas rodadas.

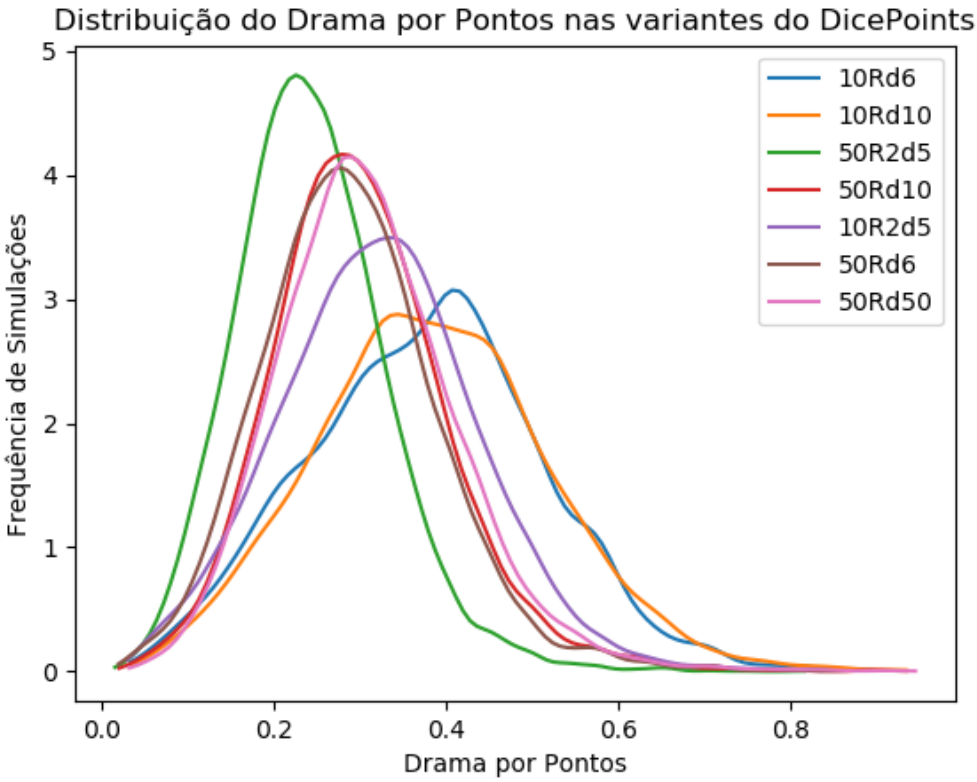

Figura 5.6 – Distribuição do Drama por Pontos no DicePoints

Em relação ao gráfico da figura 5.7, do Drama por Posição, observa-se a existência de dois padrões de dramas bem distintos. O primeiro padrão corresponde a jogos com 10 rodadas, que possui uma densidade muito alta de drama considerado baixo, na faixa de 0,5. O segundo padrão corresponde a jogos com 50 rodadas, cujo drama por frequência de simulações decresce de forma contínua.

Quanto ao gráfico da figura 5.8, do Drama por Caminho, percebe-se que todas as variantes seguem um mesmo padrão de comportamento, com uma alteração somente na amplitude de acordo com a quantidade de rodadas.

Por fim, avaliou-se o comportamento da média da métrica de Drama por Pontos de acordo com a quantidade de rodada. A figura 5.9 apresenta o comportamento da

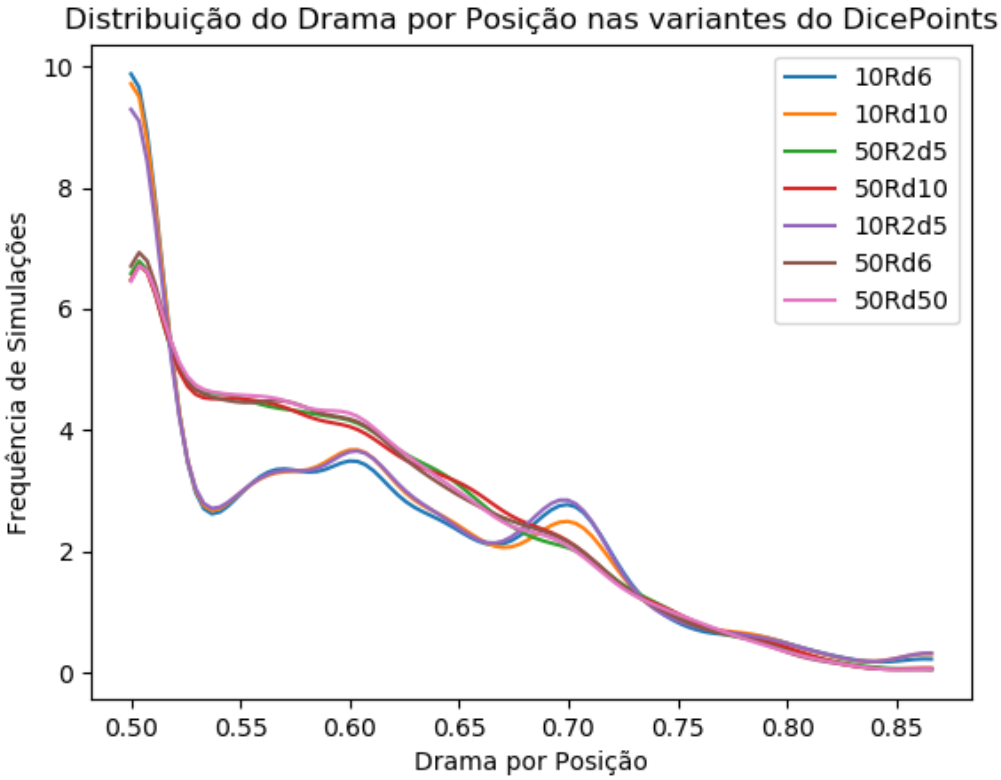

Figura 5.7 – Distribuição do Drama por Posição no DicePoints

média. É possível avaliar que a maior média ocorre quando o jogo se aproxima das 10 rodadas. Também foi possível deduzir que aumentar em demasia a quantidade de rodadas implica em uma queda na média do drama.

5.1.4 Mudança de Liderança

A tabela 5.5 apresenta a média da Mudança de Liderança no jogo DicePoints, considerando as dez mil simulações. De acordo com a tabela, todas as simulações com 10 rodadas obtiveram média maior que 0,6. Por outro lado, simulações com 50 rodadas produziram média menor que 0,6. Desta forma, é dedutível que uma quantidade menor de rodadas implica em mais mudanças de liderança no jogo DicePoints

A figura 5.10 apresenta a distribuição da métrica de Mudança de Liderança. De forma semelhante à métrica de Drama por Liderança, o comportamento da distribuição é diferenciado de acordo com a quantidade de rodadas na variante.

5.1.5 Incerteza

Os valores de incerteza para o jogo DicePoints, apresentados na tabela 5.6, coincidem com a essência do jogo. Pela pontuação ser gerada automaticamente através de lançamentos de dados com resultados aleatórios, DicePoints corresponde a um jogo

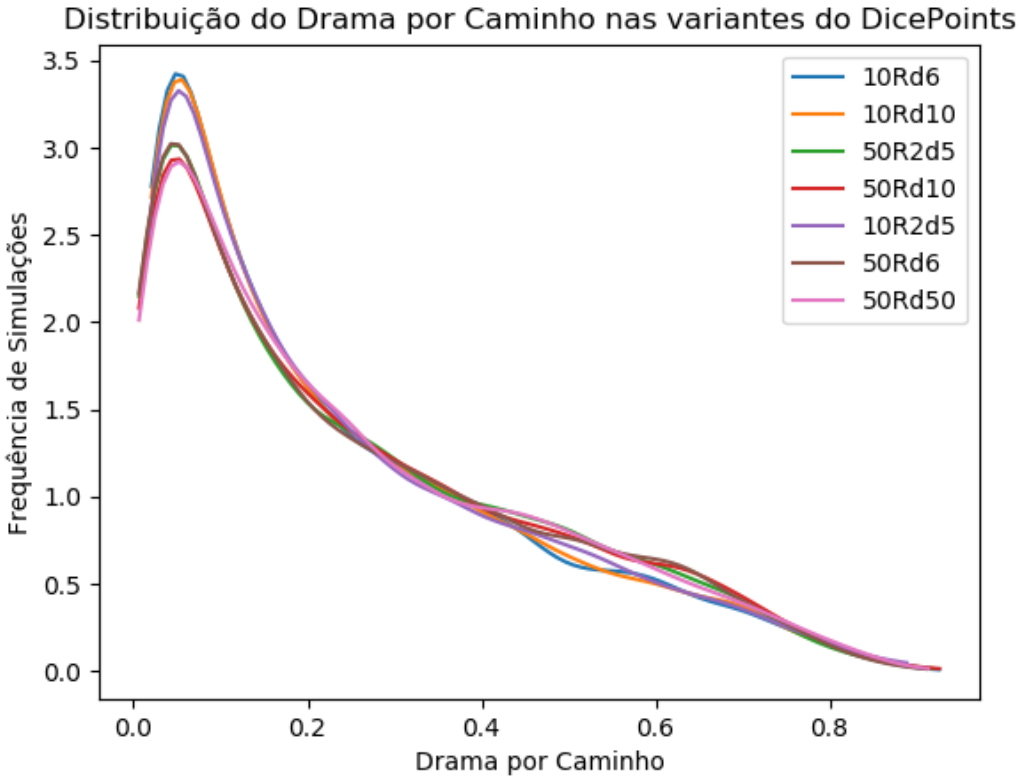

Figura 5.8 – Distribuição do Drama por Caminho no DicePoints

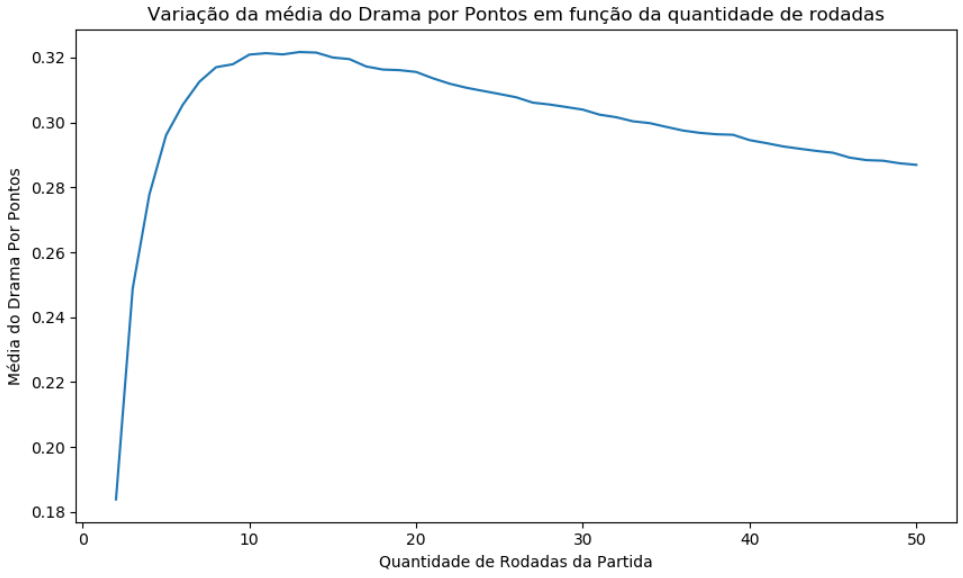

Figura 5.9 – Comportamento da média de Drama por Pontos em função da quantidade de rodadas

muito incerto. Assim, para a incerteza por entropia, a média para todas as variantes ser acima de 0,97 condiz com o fundamento do jogo. A tabela também mostra que

	Mudança de Liderança	
	Média	Desvio Padrão
10Rd6	**0,61493**	0,12894
10Rd10	**0,61661**	0,12919
10R2d5	**0,61559**	0,12953
50Rd6	**0,58315**	0,11731
50Rd10	**0,58676**	0,11685
50R2d5	**0,58466**	0,11592
50Rd50	**0,59008**	0,11661

Tabela 5.5 – Mudança de Liderança no DicePoints

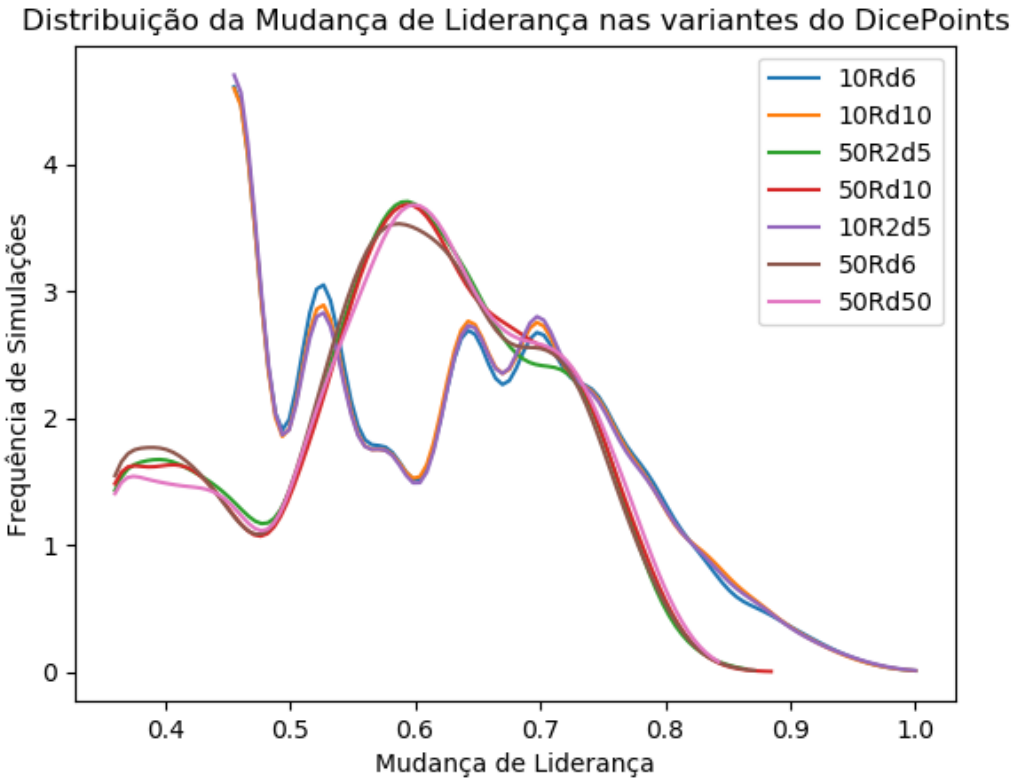

Figura 5.10 – Distribuição da Mudança de Liderança no DicePoints

o jogo torna-se ainda mais incerto nas simulações com mais rodadas, resultado que se apresenta em ambas as métricas de incerteza.

Ambos os gráficos da distribuição das métricas de Incerteza, figura 5.12 e figura 5.11, apresentam picos de frequência de simulações com valores de Incerteza altos. Em ambos os casos, quanto maior a quantidade de rodadas, maior o valor da incerteza.

	Incerteza Por Entropia		Incerteza Por PDD	
	Média	Desvio Padrão	Média	Desvio Padrão
10Rd6	**0,97718**	0,01211	**0,89162**	0,03170
10Rd10	**0,97297**	0,01479	**0,88231**	0,03532
10R2d5	**0,98996**	0,00576	**0,92846**	0,02147
50Rd6	**0,99362**	0,00301	**0,94854**	0,01366
50Rd10	**0,99258**	0,00352	**0,94457**	0,01465
50R2d5	**0,99714**	0,00140	**0,96532**	0,00920
50Rd50	**0,99100**	0,00438	**0,93946**	0,01616

Tabela 5.6 – Incerteza no DicePoints

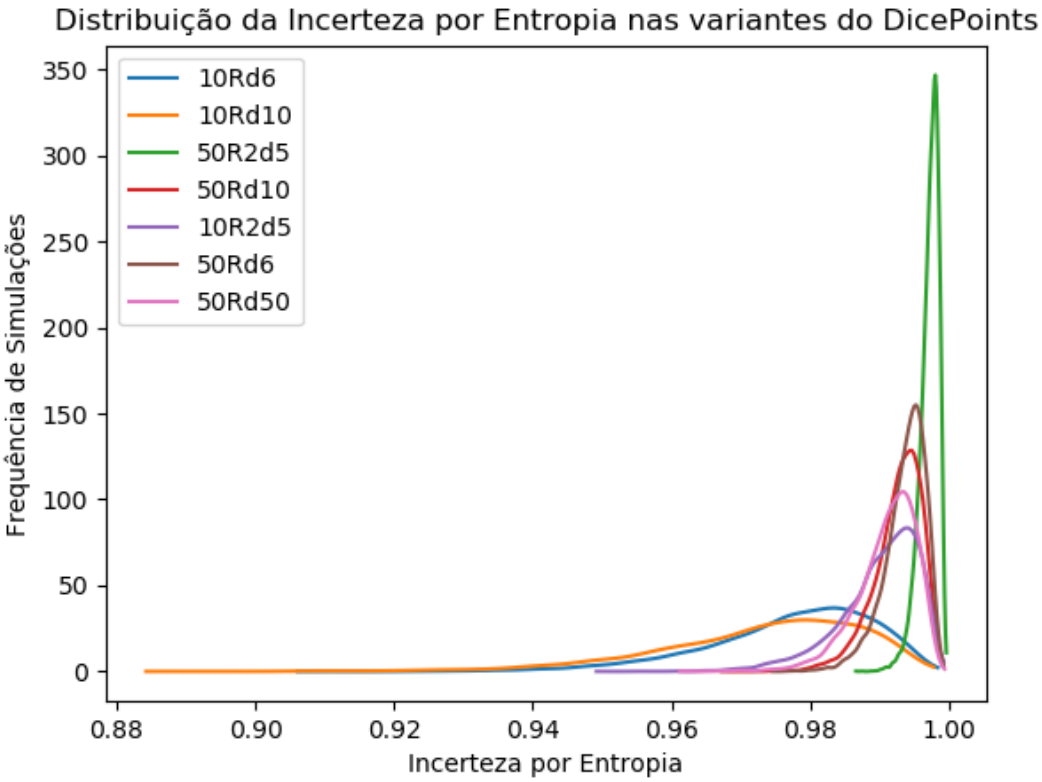

Figura 5.11 – Distribuição da Incerteza por Entropia no DicePoints

5.1.6 Conclusões

A partir dos gráficos obtidos relacionando métricas, observou-se um comportamento padrão do Drama, da Mudança de Liderança e da Incerteza de acordo com uma variável. Variar a quantidade de rodadas mostrou possuir muito mais relevância para essas métricas do que mudar a quantidade de faces. Ademais, percebeu-se o agrupamento das métricas em clusters, de acordo com a quantidade de rodada, implicando em um resultado interessante e fascinante.

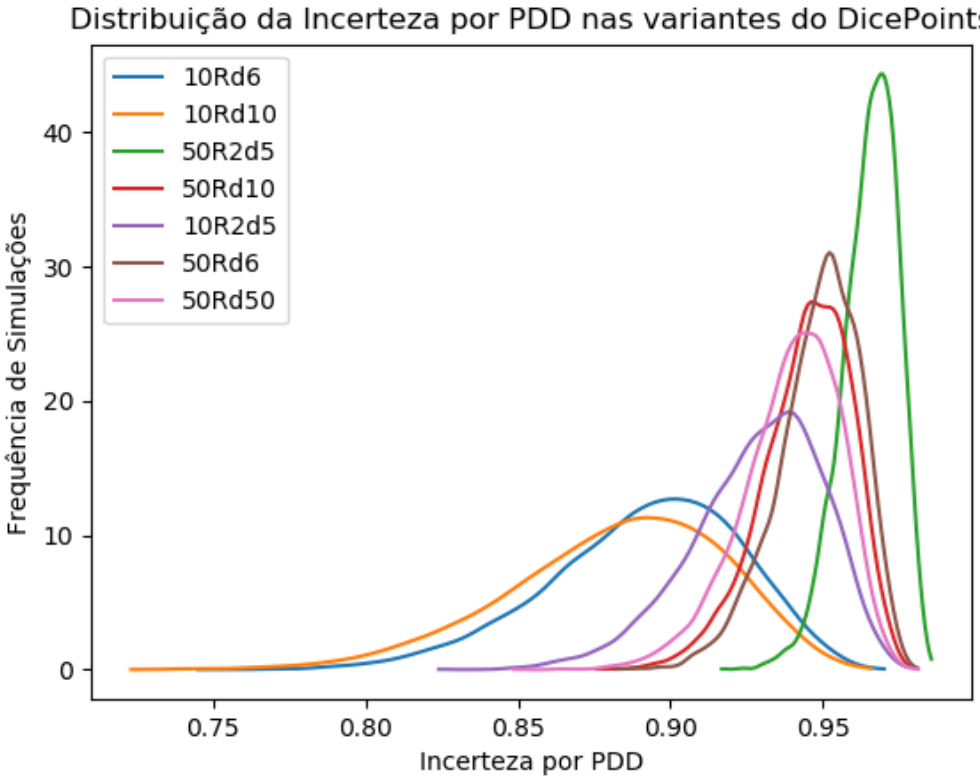

Figura 5.12 – Distribuição da Incerteza por PDD no DicePoints

5.2 ThrowDiceMatchCard

ThrowDiceMatchCard é um jogo que possui como característica a baixa pontuação. Pelos jogadores só pontuarem ao combinar o valor da carta com o valor do dado, a pontuação do jogo é escassa. Ademais, jogadores aleatórios foram utilizados para realizar a simulação, aumentando ainda mais a dificuldade em pontuar. Desta forma, com a pontuação dos jogadores começando em 0, foram realizadas simulações em que a melhor pontuação foi 0. Isso implicou em um problema, visto que as métricas implementas na *framework* de avaliação não aceitam como melhor pontuação o valor 0. Além disso, um jogo que possui como melhor pontuação o valor 0 é conceitualmente desinteressante. A descrição do jogo foi assim adaptada para fazer a pontuação começar do valor 1.

Com o objetivo de observar a variação da métrica de acordo com a quantidade de rodadas, duas variantes do jogo foram descritas. O número de faces do dado foi fixado em 5, assim como os valores das cartas variam de 1 a 5. Foi variada a quantidade de cartas recebidas pelos jogadores. Na primeira variante, cada um dos quatro jogadores recebe 5 cartas, dentre 25 possíveis, e joga 5 rodadas. Na segunda variante, cada um dos jogadores recebe 20 cartas, dentre 100 possíveis, e joga 20

rodadas. Desta forma, denominou-se as variantes de 5 Rodadas e 20 Rodadas, respectivamente.

A tabela 5.7 apresenta os valores de métrica obtidos para as duas variantes. É importante notar que a quantidade de simulações com drama e mudança de liderança aumenta muito de acordo com a quantidade de rodadas. Assim, um designer pode escolher se prefere um jogo em que o Drama ocorre ou se prefere um jogo com pouco Drama. Também é possível observar que os valores da métrica de incerteza não variam muito, ambos os jogos são muito incertos e a quantidade de rodadas não influencia nesse critério.

	5 Rodadas			20 Rodadas		
	Valor > 0	Média	Desvio Padrão	Valor > 0	Média	Desvio Padrão
Drama por Pontos	3656	0,55032	0,19064	7257	0,4389	0,1928
Drama por Posição	6285	0,60840	0,11744	8758	0,5984	0,0933
Drama por Caminho	5106	0,21184	0,19937	8758	0,2407	0,2070
Mudança de Liderança	6285	0,61567	0,10356	8758	0,5550	0,1177
Incerteza por Entropia	10000	0,95955	0,02004	10000	0,9544	0,0241
Incerteza por PDD	10000	0,85135	0,05254	10000	0,8352	0,0477

Tabela 5.7 – Resultado das métricas para variantes do ThrowDiceMatchCard

Para entender melhor o comportamento das métricas, gráficos demonstrando a distribuição dos valores da métrica foram concebidos para cada uma das variantes. A figura 5.13 apresenta os gráficos das métricas de Drama para as duas variantes, de 5 rodadas e de 20 rodadas, enquanto a figura 5.14 compara os gráficos das métricas de Mudança de Liderança e de Incerteza para ambas as variantes. É possível observar que, pela variante de 20 rodadas possuir mais valores de dramas, o drama se comporta de forma a ser mais distribuído e homogêneo de acordo com a quantidade de rodadas. O mesmo comportamento se repete para todas as métricas. Este efeito torna claro a importância de um jogo que possua muitas rodadas e onde os jogadores pontuem bastante.

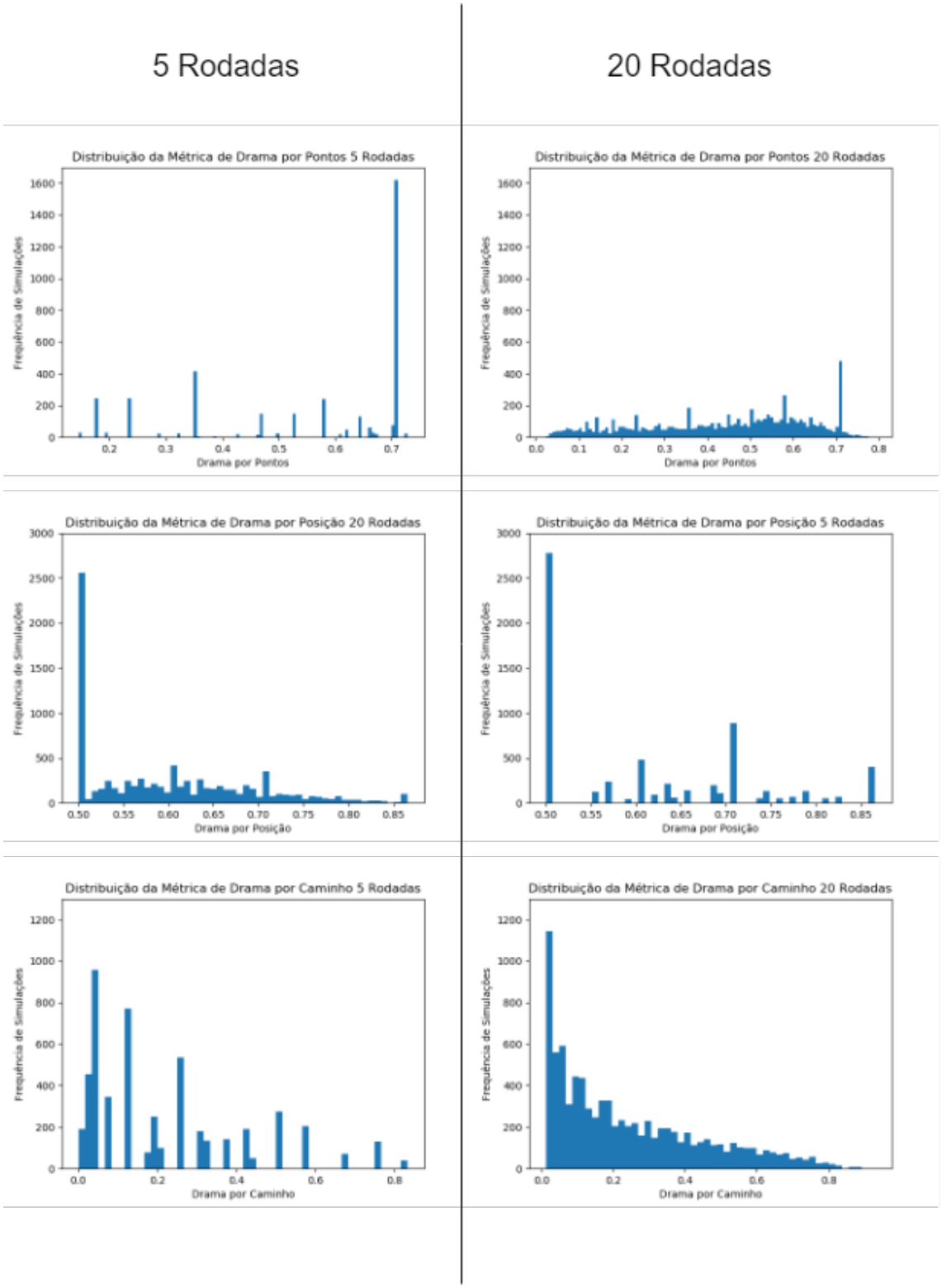

Figura 5.13 – Distribuições dos valores das métricas de Drama em jogos de 5 e de 20 rodadas do ThrowDiceMatchCard

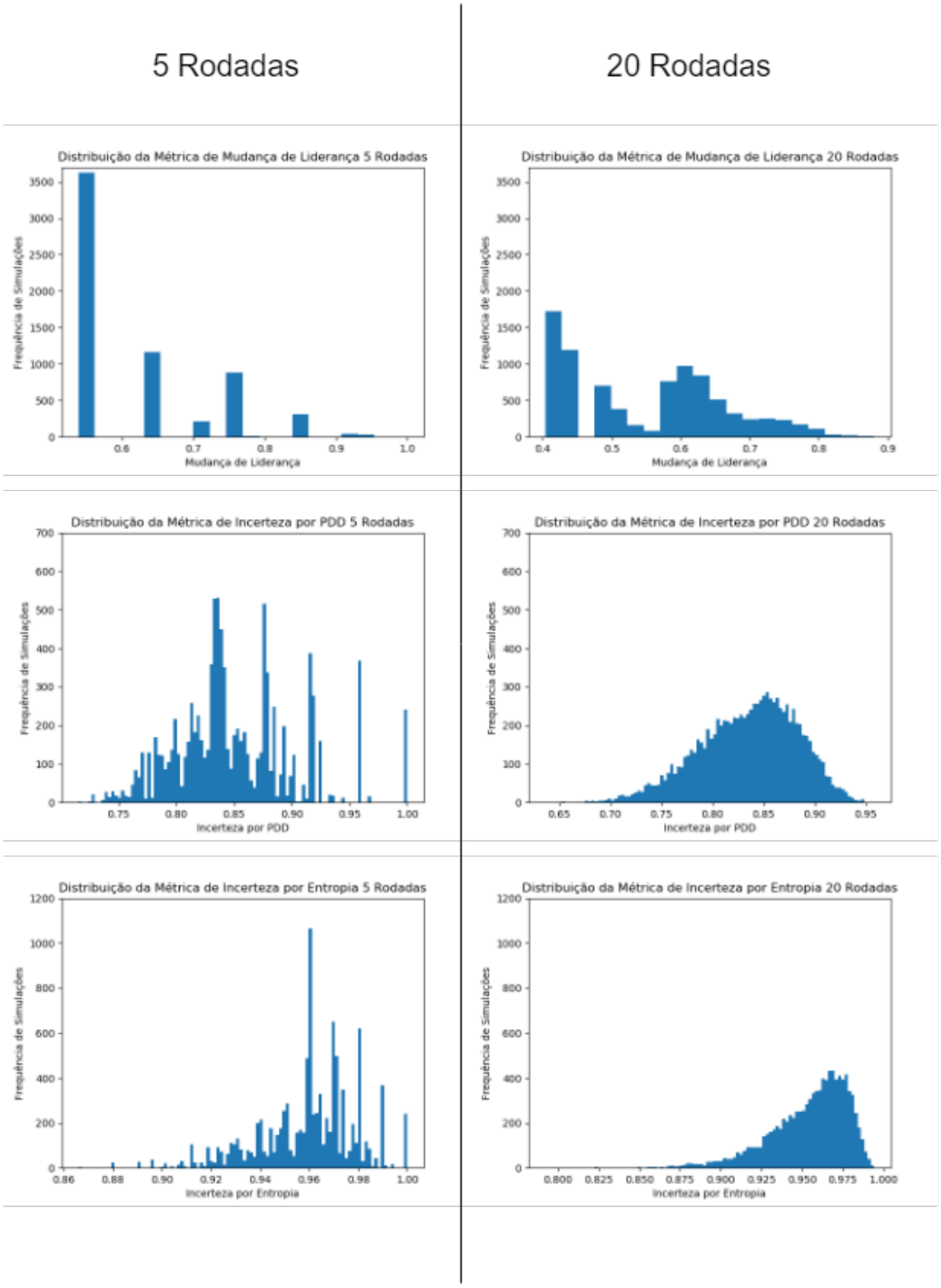

Figura 5.14 – Distribuições dos valores das métricas de Mudança de Liderança e Incerteza em jogos de 5 e de 20 rodadas do ThrowDiceMatchCard

Capítulo 6

Conclusão e Trabalhos Futuros

Este capítulo apresenta as conclusões do trabalho. Assim, na primeira sessão ele apresenta as contribuições realizadas com o intuito de resolver o problema de pesquisa e as questões de pesquisa. Por fim, na segunda sessão, os trabalhos futuros que podem ser elaborados com base na contribuição realizada são abordados.

6.1 Conclusão

O problema da pesquisa abordado neste trabalho é de investigar a possibilidade de avaliação automática de descrições de jogos de cartas e dados através de um *framework* de avaliação de estética.

A investigação do problema começou pela busca dos componentes de base da estrutura de avaliação. Foi realizada uma revisão desses componentes, compreendendo linguagem de descrição de jogos, simuladores e por fim um avaliador de jogos. Assim, uma estrutura foi concebida para responder as questões de pesquisa.

Uma linguagem de jogos de cartas foi estendida para permitir a criação e o lançamento de dados, aumentando a gama de jogos que podem ser descritos e fornecendo um elemento de aleatoriedade importante para o designer de jogos. Essa extensão cunhou-se de RECYCLEDICE. Assim, duas descrições de jogos foram descritas nesta linguagem.

Uma ferramenta capaz de interpretar e simular a linguagem de jogos de cartas foi adaptada para ser apta a ler a nova linguagem concebida, podendo assim interpretar e simular jogos possuindo o componente de dados. Esta adaptação foi chamada de GAMESTOCK.

Ademais, o simulador também foi ajustado para permitir o salvamento de pontuação a cada turno. Desta forma, foram geradas estatísticas de turno dos jogadores no formato de turnos indicado pela ferramenta de avaliação para os dois jogos descritos.

Uma nova modelagem de jogos foi integrada a um *framework* de avaliação de estéticas baseado em turnos, permitindo desta forma que as simulações geradas

anteriormente fossem inseridas na *framework* para que fossem avaliadas a partir de métricas de estética.

Para cada jogo, variantes foram criadas e suas simulações foram fornecidas como entrada para a ferramenta. A partir delas, avaliou-se o comportamento de partidas, assim como a distribuição e o comportamento de cada métrica de estética.

Para o jogo DicePoints, as métricas de Drama por Pontos, Drama por Posição e Drama por Caminho foram comparadas em sete variantes, e observou-se características interessantes para cada variante que influenciam no valor do Drama, como quantidade de rounds e quantidade de faces de um dado. A avaliação da métrica de Mudança de Liderança indicou que uma quantidade maior de rodadas implica em menos mudanças de liderança. Observou-se também que as métricas de Incerteza por Entropia e Incerteza por PDD foram condizentes com o resultado esperado.

Em relação ao jogo ThrowDiceMatchCard, avaliou-se as mesma métricas, aplicada a duas variantes do jogo. Obteve-se como resultado que partidas com mais rodadas implicam em mais presença de Drama, além da incerteza em ambas as variantes ser sempre alta.

Assim, além de validar as métricas do *framework*, verificou-se o funcionamento da estrutura para avaliação. Os resultados demonstram, desta forma, que é possível avaliar automaticamente jogos descritos em uma linguagem de cartas e dados.

Assim, as principais contribuições neste trabalho foram:

- Concepção de uma estrutura para avaliação automatizada de jogos de dados e cartas.
- A linguagem de dados e cartas RECYCLEDICE.
- O interpretador e simulador GAMESTOCK.
- A adaptação do *framework* de avaliação para permitir simulações.

6.2 Trabalhos Futuros

A partir das ideias desenvolvidas neste trabalho, novos projetos podem ser desenvolvidos.

Métricas que se utilizam de aleatoriedade podem ser desenvolvidas e embutidas no *framework* de avaliação para verificar a relevância desse elemento em determinados jogos de cartas e dados.

Diferentes agentes inteligentes podem ser desenvolvidos com objetivos distintos e embutidos no simulador. Desta forma, o designer poderia escolher o jogo no qual as estéticas mais se assemelham ao perfil de jogador desejado.

Para fechar o ciclo de design de jogos automatizado e desenvolver novos jogos, pode ser realizada a criação de um gerador automático de jogos, que faça uso das métricas de estética como função de avaliação para a busca de variantes.

Apesar de toda a automatização, a mente humana ainda não é possível de ser replicada. Portanto, é necessário reafirmar que o processo de criação de um jogo não é possível sem uma validação final pelo ser humano. Um processo final de *playtest* com humanos pode assim ser adicionado nessa estrutura, com o objetivo de realizar uma avaliação humana das métricas e, por fim, obter um jogo finalizado.

Referências Bibliográficas

AHO, A. V., 2003, *Compilers: principles, techniques and tools (for Anna University), 2/e*. Pearson Education India.

ALTHÖFER, I., 2003, "Computer-aided game inventing", *Friedrich Schiller Universität, Jena, Germany, Tech. Rep.*

AZARIA, Y., SIPPER, M., 2005, "GP-Gammon: Using genetic programming to evolve backgammon players". In: *European Conference on Genetic Programming*, pp. 132–142. Springer.

BELL, C., GOADRICH, M., 2016, "Automated Playtesting with RECYCLEd CARDSTOCK", *Game & Puzzle Design*, v. vol. 2, n. no. 1, pp. pp. 71–83.

BJORNSSON, Y., FINNSSON, H., 2009, "CadiaPlayer: A Simulation-Based General Game Player", *IEEE Transactions on Computational Intelligence and AI in Games*, v. 1, n. 1 (mar.), pp. 4–15. ISSN: 1943-068X. doi: 10.1109/TCIAIG.2009.2018702.

BOWLING, M., BURCH, N., JOHANSON, M., et al., 2015, "Heads-up limit hold'em poker is solved", *Science*, v. 347, n. 6218, pp. 145–149.

BROWNE, C. B., 2008, *Automatic generation and evaluation of recombination games*. PhD Thesis, Queensland University of Technology.

CHASLOT, G., BAKKES, S., SZITA, I., et al., 2008, "Monte-Carlo Tree Search: A New Framework for Game AI." In: *AIIDE*.

CORREIA, J. P. C., 2013, "PGDL: Sistema para definição genérica de jogos de Poker", .

DE BONDT, M., 2012, "Solving Mahjong Solitaire boards with peeking", *arXiv preprint arXiv:1203.6559*.

EBNER, M., LEVINE, J., LUCAS, S. M., et al., 2013, "Towards a video game description language", .

FOGEL, D. B., 1993, "Using evolutionary programming to create neural networks that are capable of playing tic-tac-toe". In: *International Conference on Neural Networks*, pp. 875–880. IEEE Press San Francisco, CA.

FONT, J. M., MAHLMANN, T., MANRIQUE, D., et al., 2013, "A card game description language". In: *European Conference on the Applications of Evolutionary Computation*, pp. 254–263. Springer.

FULLERTON, T., 2008, *Game design workshop: a playcentric approach to creating innovative games*. CRC press.

GENESERETH, M., LOVE, N., PELL, B., 2005, "General game playing: Overview of the AAAI competition", *AI magazine*, v. 26, n. 2, pp. 62.

HSU, F.-H., 1999, "IBM's Deep Blue Chess grandmaster chips", *IEEE Micro*, v. 19, n. 2 (mar.), pp. 70–81. ISSN: 0272-1732. doi: 10.1109/40.755469.

HUNICKE, R., LEBLANC, M., ZUBEK, R., 2004, "MDA: A formal approach to game design and game research". In: *Proceedings of the AAAI Workshop on Challenges in Game AI*, v. 4, p. 1722.

ISAKSEN, A., ISMAIL, M., BRAMS, S. J., et al., 2015, "Catch-up: A game in which the lead alternates", *Game & Puzzle Design*, v. 1, pp. 38–49.

ISAKSEN, A., HOLMGA, C., TOGELIUS, J., et al., 2016, "Characterising Score Distributions in Dice Games", *Game and Puzzle Design*, v. 2, n. 1, pp. 14.

KRITZ, J., MANGELI, E., XEXÉO, G., 2017, "Building an Ontology of Boardgame Mechanics based on the BoardGameGeek Database and the MDA Framework". In: *XVI SBGames*.

LOVE, N., HINRICHS, T., HALEY, D., et al., 2008, *General game playing: Game description language specification*. Stanford Logic Group Computer Science Department Stanford University, Technical Report LG-2006-01.

MAHLMANN, T., TOGELIUS, J., YANNAKAKIS, G. N., 2011, "Modelling and evaluation of complex scenarios with the Strategy Game Description Language". pp. 174–181. IEEE, aug. ISBN: 978-1-4577-0010-1. doi: 10.1109/CIG.2011.6032004. `http://ieeexplore.ieee.org/document/6032004/`.

MANGELI, E. F. M., 2016, *An Aesthetic Metric for Multiplayer Turn-based Games*. Master Thesis, Universidade Federal do Rio de Janeiro.

PARR, T., 2013, *The definitive ANTLR 4 reference*. Pragmatic Bookshelf.

SCHAEFFER, J., 2008, *One jump ahead: computer perfection at checkers*. Springer Science & Business Media.

SCHAEFFER, J., BURCH, N., BJÖRNSSON, Y., et al., 2007, "Checkers is solved", *science*, v. 317, n. 5844, pp. 1518–1522.

TESAURO, G., 2002, "Programming backgammon using self-teaching neural nets", *Artificial Intelligence*, v. 134, n. 1-2, pp. 181–199.

THIELSCHER, M., 2011a, "GDL-II", *KI - Künstliche Intelligenz*, v. 25, n. 1 (mar.), pp. 63–66. ISSN: 0933-1875, 1610-1987. doi: 10.1007/s13218-010-0076-5. https://link.springer.com/article/10.1007/s13218-010-0076-5 .

THIELSCHER, M., 2016, "GDL-III: A Proposal to Extend the Game Description Language to General Epistemic Games." In: *ECAI*, v. 16, pp. 1630–1631.

THIELSCHER, M., 2011b, "The general game playing description language is universal". In: *IJCAI Proceedings-International Joint Conference on Artificial Intelligence*, v. 22, p. 1107, b.

THOMPSON, J. M., 2000. "Defining the Abstract". jul. http://www.thegamesjournal.com/articles/DefiningtheAbstract.shtml .

VAN ECK, N. J., VAN WEZEL, M., 2008, "Application of reinforcement learning to the game of Othello", *Computers & Operations Research*, v. 35, n. 6, pp. 1999–2017.

WIJMAN, T., 2018. "Global Games Market Revenues 2018 | Per Region & Segment". https://newzoo.com/insights/articles/global-games-market-reaches-137-9-billion-in-2018-mobile-games-take-half/ .

Apêndice A

Descrições de Jogos

A.1 Agram em RECYCLE

Na linha 2, a variável com quantidade de jogadores é declarada. A partir da linha 3 a 17, a criação dos jogadores, dos times e do *deck* é realizada. Ao *deck* são adicionados dois conjuntos. O primeiro conjunto de cartas corresponde a 32 cartas, dividas em 4 naipes com valores de 3 a 10. O primeiro conjunto é instanciado nas linhas 6 a 11. O segundo conjunto, criado nas linhas 12 a , corresponde a criação das cartas de ás, nos outros três naipes.

Da linha 18 a 24, as cartas são embaralhadas com o *shuffle* e cada jogador recebe 6 carta através da regra *move*. Na linha 25, um novo *stage* se inicia, que termina quando todos os jogadores esvaziarem suas mãos. Outro *stage* é na linha 28, que itera sobre todos os jogadores, terminando quando todos tiverem jogado uma carta na mesa. Da linha 31 a 62, descreve as opções do jogador atual, que pode realizar três escolhas.

- Linha 32 até 42 : Se o jogador não possui carta para seguir o naipe, ele pode jogar qualquer carta de sua mão.
- Linha 44 até 52 : Se o jogador não for o primeiro a jogar e possui alguma carta do naipe da mesa, o jogador é obrigado a jogar uma carta seguindo o naipe.
- Linha 53 até 62 : Se o jogador é o primeiro da rodada, ele pode jogar qualquer carta. O naipe escolhido é memorizado pelo jogo.

Após o fim de uma rodada, tem-se um conjunto de ações realizadas pelo computador. Entre as linhas 64 e 76, a precedência das cartas é realizada, indicando o valor de cada carta. A linha 77 é responsável por apagar da memória o vencedor da rodada anterior. Nas linhas 78 a 81, o jogador inicial da próxima rodada é indicado como aquele que possui a maior carta do naipe correto na mesa. Nas linhas 82 a 84, a mesa de cada jogador é esvaziada. Entre as linhas 85 a 87, caso as mãos dos

jogadores estejam vazias, o vencedor da última rodada recebe um ponto. O vencedor do jogo é definido na linha 90, através da regra *scoring*.

```
1  (game
2   (declare 4 'NUMP)
3   (setup
4    (create players 'NUMP)
5    (create teams (0) (1) (2) (3))
6    (create deck (game iloc STOCK)
7     (deck
8      (RANK (THREE, FOUR, FIVE, SIX, SEVEN, EIGHT, NINE, TEN))
9      (COLOR (RED (SUIT (HEARTS, DIAMONDS)))
10            (BLACK (SUIT (SPADES, CLUBS))))
11    ))
12    (create deck (game iloc STOCK)
13     (deck
14      (RANK (ACE))
15      (COLOR (RED (SUIT (HEARTS, DIAMONDS)))
16            (BLACK (SUIT (CLUBS))))
17  )))
18  (do (
19   (shuffle (game iloc STOCK))
20   (all player 'P
21     (repeat 6
22      (move (top (game iloc STOCK))
23            (top ('P iloc HAND)))))
24  ))
25  (stage player
26   (end (all player 'P
27              (== (size ('P iloc HAND)) 0)))
28   (stage player
29      (end (all player 'P
30             (> (size ('P vloc TRICK)) 0)))
31      (choice (
32       ((and
33         (== (size (game mem LEAD)) 1)
34         (== (size (filter ((current player) iloc HAND) 'C
35              (== (cardatt SUIT 'C)
36                  (cardatt SUIT (top (game mem LEAD)))
37              ))) 0)
38        )
39        (any ((current player) iloc HAND) 'AC
```

```
            (move 'AC
                (top ((current player) vloc TRICK))))
      )

      (any
        (filter ((current player) iloc HAND) 'T
            (== (cardatt SUIT 'T)
                (cardatt SUIT (top (game mem LEAD)))
            )
        ) 'C
        ((== (size (game mem LEAD)) 1)
             (move 'C (top ((current player) vloc TRICK))))
      )
      ((== (size (game mem LEAD)) 0)
          (any ((current player) iloc HAND) 'AC
           (do (
               (move 'AC
                     (top ((current player) vloc TRICK)))
               (remember (top ((current player) vloc TRICK))
                         (top (game mem LEAD)))
           ))
           )
 ))))
 (do (
  (put points 'PRECEDENCE (
       ((SUIT (cardatt SUIT (top (game mem LEAD)))) 100)
       ((RANK (ACE)) 14)
       ((RANK (TEN)) 10)
       ((RANK (NINE)) 9)
       ((RANK (EIGHT)) 8)
       ((RANK (SEVEN)) 7)
       ((RANK (SIX)) 6)
       ((RANK (FIVE)) 5)
       ((RANK (FOUR)) 4)
       ((RANK (THREE)) 3)
       )
  )
  (forget (top (game mem LEAD)))
  (cycle next
   (owner (max (union (all player 'P ('P vloc TRICK)))
        using 'PRECEDENCE))
```

```
81    )
82    (all player 'P
83            (move (top ('P vloc TRICK))
84    (top (game vloc DISCARD))))
85    ((all player 'P
86       (== (size ('P iloc HAND)) 0))
87           (inc ((next player) sto SCORE) 1))
88   ))
89  )
90  (scoring max (((current player) player) sto SCORE))
91 )
```

Código A.1 – Descrição do jogo AGRAM em RECYCLE

A.2 ThrowDiceMatchCard em RECYCLEDICE

```
1  (game
2   (declare 4 'NUMP)
3   (setup
4    (create players 'NUMP)
5    (create teams (0) (1) (2) (3))
6    (repeat 5 (create deck (game iloc STOCK) (deck (VALUE (ONE,
      TWO, THREE, FOUR, FIVE)))))
7    (create dicestorage 'GAMEDICE (5))
8   )
9   (do
10   (
11    (put points 'PRECEDENCE
12     (
13      ((VALUE (FIVE)) 5)
14      ((VALUE (FOUR)) 4)
15      ((VALUE (THREE)) 3)
16      ((VALUE (TWO)) 2)
17      ((VALUE (ONE)) 1)
18    ))
19    (shuffle (game iloc STOCK))
20    (all player 'P
21   (do (
22      (repeat 20 (move (top (game iloc STOCK))
23                               (top ('P iloc HAND))))
24      (set ((current player) sto SCORE) 1)
```

```
 ))
  )
  (set (game sto ROUNDS) 0)
))
(stage player
 (end (all player 'P
            (== (size ('P iloc HAND)) 0)))
 (do
  (
   (throwalldice 'GAMEDICE)
 ))
 (stage player
  (end (all player 'P
               (> (size ('P vloc TRICK)) 0)))
  (choice
   (
    (do
     (
      (any ((current player) iloc HAND) 'AC
           (move 'AC (top ((current player) vloc TRICK))))
  ))))
  (do
   (
    (
     (==
      (sum ((current player) vloc TRICK) using 'PRECEDENCE)
      (dievalue 'GAMEDICE)
     )
      (inc ((current player) sto SCORE) 1)
    )
)))
(do
 (
  (all player 'P
       (move (top ('P vloc TRICK))
             (top (game vloc DISCARD))))
  (cycle current previous)
  (inc (game sto ROUNDS) 1)
  (saveturnstats SCORE roundnumber ROUNDS)
)))
(scoring max ((current player) sto SCORE))
```

```
66 )
```

Código A.2 – Descrição do jogo *ThrowDiceMatchCard* em RECYCLEDICE