UNIVERSIDADE ESTADUAL DA PARAÍBA
CAMPUS I - CAMPINA GRANDE/PB
CENTRO DE CIÊNCIAS E TECNOLOGIA
DEPARTAMENTO DE COMPUTAÇÃO
CURSO DE BACHARELADO EM CIÊNCIAS DA COMPUTAÇÃO

JOAQUIM ANÍBAL ROCHA NETO

PROGRAMAÇÃO FUNCIONAL E TEORIA DOS JOGOS - IMPLEMENTANDO O JOGO CORONEL BLOTTO

CAMPINA GRANDE
2019

JOAQUIM ANÍBAL ROCHA NETO

PROGRAMAÇÃO FUNCIONAL E TEORIA DOS JOGOS - IMPLEMENTANDO O JOGO CORONEL BLOTTO

Trabalho de Conclusão de Curso apresentado ao Curso de Bacharelado em Ciências da Computação da Universidade Estadual da Paraíba, como requisito parcial à obtenção do título de Bacharel em Ciências da Computação.

Área de concentração: Linguagens de Programação.

Orientador: Prof. Me. Edson Holanda Cavalcante Júnior

CAMPINA GRANDE
2019

R672p Rocha Neto, Joaquim Anibal.

Programação funcional e teoria dos jogos [manuscrito] : implementando o jogo Coronel Blotto / Joaquim Anibal Rocha Neto. - 2019.

63 p. : il. colorido.

Digitado.

Trabalho de Conclusão de Curso (Graduação em Computação) - Universidade Estadual da Paraíba, Centro de Ciências e Tecnologia , 2019.

"Orientação : Prof. Me. Edson Holanda Cavalcante Júnior , Coordenação do Curso de Computação - CCT."

1. Teoria dos Jogos. 2. Programação funcional. 3. Haskell. 4. Coronel Blotto. I. Título

21. ed. CDD 005.13

Elaborada por Giulianne M. Pereira - CRB - 15/714 BC/UEPB

JOAQUIM ANÍBAL ROCHA NETO

PROGRAMAÇÃO FUNCIONAL E TEORIA DOS JOGOS - IMPLEMENTANDO O JOGO CORONEL BLOTTO

Trabalho de Conclusão de Curso de Graduação em Ciência da Computação da Universidade Estadual da Paraíba, como requisito à obtenção do título de Bacharel em Ciência da Computação.

Aprovada em 10 de Dezembro de 2019.

Prof. MSc. Edson Holanda Cavalcante Júnior (DC - UEPB)
Orientador(a)

Prof. Dr. Wellington Candeia De Araújo (DC - UEPB)
Examinador(a)

Prof. Dr. Alysson Filgueira Milanez (DC - UEPB)
Examinador(a)

A Deus por tudo e à minha família pelo enorme apoio, dedico com carinho.

AGRADECIMENTOS

Agradeço primeiramente a Deus, por estar concluindo mais uma fase em minha vida, sem o seu cuidado nada seria possível.

Ao meu orientador, professor Me. Edson Holanda Cavalcante Júnior, pela inestimável dedicação com a qual me orientou nesta jornada.

À UEPB, pela grande oportunidade que a mim foi dada.

Aos meu pais, José de Deus e Maria Lucimar, por serem os meus maiores incentivadores em qualquer momento, pelo imensurável apoio nesta caminhada, exemplos de perseverança.

À minha irmã Júlia, pela confiança que sempre depositou em mim.

Aos meus que já se foram, mas que carrego comigo para sempre, pelos exemplos deixados e que sigo com orgulho a partir de eternas lembranças.

À minha avó, tios, primos, por participarem dos momentos marcantes de minha vida.

À minha namorada Nayara, pela enorme compreensão e carinho ao longo desse tempo.

À minha colega Mariana, pela grande contribuição nesta pesquisa.

Aos professores, que contribuíram ao longo desta longa caminhada para o meu crescimento, não só nos estudos, mas também como pessoa.

Aos colegas e amigos que o curso me proporcionou, por compartilhar os momentos de alegria e tristeza no decorrer dessa jornada, juntos em busca do mesmo objetivo.

Aos meus amigos, pelas palavras de incentivo, não me deixando desacreditar do foco, pela ajuda nas mais diversas horas do dia.

Aos motoristas e demais funcionários da prefeitura, que contribuíram de forma profissional e amiga nas idas e vindas à UEPB.

Aos funcionários da UEPB, pela presteza e atenção com a qual realizam suas tarefas.

Meus sinceros agradecimentos a todos!

“A fé na vitória tem que ser inabalável”
(Dexter)

RESUMO

O presente trabalho tem como objetivo principal apresentar o poder da programação funcional por meio da implementação do jogo Coronel Blotto com a linguagem de programação Haskell, que é uma linguagem puramente funcional. No jogo, o foco principal será abordar alguns conceitos pertinentes à Teoria dos Jogos, área que tem muita importância não só para a Computação, como também para a Economia, Biologia, Política, etc. Isso será feito da forma mais simples, direta e prática possível, sendo utilizado o jogo Coronel Blotto, que é amplamente estudado na Teoria dos Jogos. A abordagem da parte teórica da Teoria dos Jogos é realizada de forma dinâmica, com o jogador sendo acompanhado e recebendo orientações de como são empregados os conceitos, com o intuito de oferecer e proporcionar as próprias percepções ao jogador, buscando, assim, diversificar a maneira como os conceitos são repassados e demonstrando na prática como funciona. Tanto a parte lógica como a parte gráfica e de interação com o usuário foram implementadas utilizando Haskell, uma das principais linguagens que seguem o paradigma de programação funcional de forma pura, aproveitando esse projeto para comprovar o poder proporcionado por ela e que é desconhecido por muitos programadores. Após uma exposição simplista sobre o jogo, serão apresentados alguns dos pontos principais desse paradigma, seus princípios, suas vantagens, como também características da linguagem Haskell a fim de situar o leitor. Serão abordados também o processo de desenvolvimento e as suas particularidades, enfatizando questões relacionadas com a programação funcional. Ao fim será apresentada a versão desenvolvida do jogo, comprovando as características apresentadas, juntamente com as possíveis linhas de melhoria que podem ser seguidas após esse trabalho.

Palavras-Chave: Teoria dos Jogos. Programação Funcional. Haskell. Coronel Blotto.

ABSTRACT

This work aims to present the power of functional programming, through the implementation of the game Colonel Blotto with the Haskell programming language, which is a purely functional language. In the game, the main focus will be to address some concepts relevant to Game Theory, an area that is very important not only for computing, but also for economics, biology, politics, etc. This will be as simple, straightforward and practical as possible, using the game Colonel Blotto, which is widely studied in Game Theory. The approach of theoretical part of Game Theory is performed in a dynamic way, with the player being accompanied and receiving guidance on how the concepts are employed, in order to offer and provide the player with their own perceptions. Seeking to diversify the way concepts are passed on and demonstrating in practice how it works. Both the logical and graphical parts of user interaction were implemented using Haskell, one of the main languages that follow the functional programming paradigm in a pure way, taking advantage of this project to prove the power provided by it, and which is unknown by too many programmers. After a simplistic exposition on the game, some of the main points of this paradigm, its principles, its advantages, as well as characteristics of the Haskell language will be presented, in order to situate the reader. It will also address the development process and its particularities, emphasizing issues related to functional programming. At the end will be presented the developed version of the game, proving the characteristics presented, along with the possible improvement lines that can be followed after this work.

Keywords: Game Theory. Functional Programming. Haskell. Colonel Blotto.

LISTA DE FIGURAS

LISTA DE QUADROS

SUMÁRIO

1 INTRODUÇÃO

O presente trabalho está inserido em um contexto que abrange algumas áreas de estudo na Computação, desde áreas mais teóricas, como a Teoria dos Jogos, até áreas mais práticas como a linguagem de programação Haskell.

O objetivo principal deste trabalho é demonstrar o poder da programação funcional utilizando a linguagem de programação Haskell, por meio da implementação de uma versão do jogo Coronel Blotto, jogo muito utilizado para estudos na área da Teoria dos Jogos, em diversas áreas de aplicação, como na política, economia, publicidade, esportes, etc.

Um objetivo secundário seria desmistificar um certo "preconceito" que se tem por muitos programadores à respeito da programação funcional, principalmente com relação à interação com o usuário.

Inicialmente são apresentados os materiais estudados para a produção do trabalho, como por exemplo, Teoria dos Jogos, programação funcional, assuntos envolvidos na produção do "artefato" principal do trabalho, o jogo.

É debatido minuciosamente o processo de desenvolvimento, até chegar ao produto, enfatizando o quão simples é a produção de programas com robustez utilizando até mesmo interface gráfica para interação com o usuário, utilizando uma linguagem de programação funcional.

O trabalho está dividido por seções a fim de melhorar a separação e a organização dos assuntos, com o intuito de facilitar o entendimento por parte do leitor. É realizada uma abordagem de toda a teoria empregada no trabalho, iniciando com a explanação do tema Teoria dos Jogos, abordando um pouco de história, desde os primeiros registros relacionados com o tema até às definições mais recentes, apresentando as principais definições dessa área e o jogo com suas características e elementos presentes.

Também serão apresentadas as diversas formas de classificações dos jogos, buscando situar o leitor de acordo com uma área de classificação e diferenciar bem para que fique claro a qual especificação o jogo pertence. São apresentados também juntamente com a definição, exemplos de jogos (são descritos no Apêndice A) que se classificam dessa maneira e o porquê de ele ser classificado assim.

Seguindo os assuntos é apresentado o paradigma de programação funcional, paradigma muito forte em diversas áreas da Computação. São discutidos alguns de seus

pontos principais, abrangendo um pouco da sua história para definir todo o contexto no qual o mesmo foi criado. Serão discutidas sua relação profunda com as funções, os pontos positivos desse paradigma, como também alguns pontos que são melhor trabalhados por outros paradigmas e as novas ideias que ele oferece ao programador.

Com a ideia de programação funcional apresentada previamente, foi discutida a linguagem Haskell, que foi utilizada para implementação do jogo trabalhado. Dentre outras, ela segue estritamente os pilares funcionais e foi criada com esse intuito. É apresentado de forma geral como ela implementa os conceitos funcionais, entre outras características da linguagem.

Após a apresentação dos conceitos e apetrechos que serão utilizados para implementação do jogo, foi discutido o jogo propriamente dito. Como nas outras seções foi abordada também um pouco da história, os conceitos da Teoria dos Jogos presentes no jogo Coronel Blotto e uma correlação das classificações apresentadas para o jogo, assim como suas justificativas. Também serão demonstradas algumas das áreas de aplicação do jogo no mundo real. Por fim, serão apresentadas as definições específicas utilizadas na implementação do jogo neste trabalho.

É apresentado o processo de implementação do jogo Coronel Blotto, buscando empregar conceitos da engenharia de software para conduzir esse processo, falando um pouco da metodologia adotada, a metodologia ágil, citando o passo a passo do que foi realizado, apresentando os artefatos que foram gerados ao decorrer e a relação com o orientador ao longo do processo.

Inicialmente foi realizada uma explanação da parte interna da aplicação (*backend*), envolvendo toda a lógica do projeto e posteriormente a respeito da implementação da interface gráfica (*frontend*) do jogo, requisito que foi inicialmente sugerido pelo orientador. Também falamos de forma abrangente sobre a biblioteca Haskell BRICK, adotada para implementação, a motivação para sua escolha, o modo de funcionamento da mesma, com detalhes de como foi implantada no projeto, servindo de exemplo para o leitor.

Finalizando o trabalho são discutidas as dificuldades enfrentadas, juntamente com a apresentação do produto que foi alcançado a partir do desenvolvimento.

2 REVISÃO DA LITERATURA

Nesta Seção serão apresentados os “pilares” que serviram de arcabouço para a produção deste trabalho, do objetivo principal. Descrevendo cada um deles em detalhes para situar o leitor sobre os assuntos envolvidos na implementação do jogo Coronel Blotto.

2.1 Teoria dos Jogos

Como foi descrito na Introdução, iniciamos a construção deste trabalho apresentando o tema que será abordado na implementação do jogo Coronel Blotto, trabalhado na Teoria dos Jogos, assunto de grande importância em diversas áreas, como por exemplo, na política, economia, biologia, publicidade, etc. Inicialmente será apresentada um pouco da sua história, desde o “nascimento”, seguido do debate de alguns de seus conceitos de forma mais aprofundada.

A Teoria dos Jogos se fundamenta em modelos matemáticos para estudar a interação entre dois ou mais indivíduos racionais, cada qual com seus próprios interesses, podendo ou não ser conflitantes, buscando escolher decisões ótimas, que otimizem a sua recompensa ao final (MYERSON, 1991; SARTINI et al., 2004; CARVALHO, 2013; RIBEIRO, 2013; FRANCEZ, 2017).

Ela tem seus primeiros registros realmente nomeados como Teoria dos Jogos antes mesmo de 1930, quando John von Neumann, um matemático húngaro e Oskar Morgenstern, economista austríaco, iniciaram estudos sobre o tema em 1928, chegando a lançar o livro *Theory of Games and Economic Behavior* em 1944, tido por muitos como o principal documento a respeito do assunto, chegando a fixar a Teoria dos Jogos como um campo de estudo (CÂMARA, 2011).

A obra acima citada apresenta e debate principalmente o tema jogos de soma zero (FIANI, 2009). Nela é relatada uma abordagem da Teoria dos Jogos voltada para a área da Economia e baseia-se em uma teoria matemática envolvendo jogos de estratégia desenvolvida por eles ao longo de anos de estudo após a identificação da relação entre a teoria econômica e a Teoria dos Jogos com alguns comportamentos idênticos (ALMEIDA, 2006).

Muito antes disso, no início do século XVIII surgiram registros, mesmo que discretos, do que no futuro chegaria a se classificar como a área da Teoria dos Jogos. James Waldegrave enviou uma correspondência datada do ano de 1713 ao seu amigo Pierre Rémond de

Montmort, essa carta continha uma análise feita por Waldegrave a respeito do jogo francês de cartas *Le Her,* onde ele demonstra a utilização de estratégia mista, envolvendo uma distribuição de probabilidade, com o intuito de fornecer uma maior chance de vitória ao jogador, independente da estratégia adversária. Essa ideia foi estudada e publicada nesse mesmo ano por Montmort e Bernoulli, um matemático suíço (BORTOLOSSI; GARBUGIO; SARTINI, 2017).

Em 1838, Cournot propôs em sua obra *Recherches sur les Principes Mathematiques de la Theorie des Richesses,* uma abordagem prematura de equilíbrio de Nash, definindo um tipo de equilíbrio entre estratégias, desenvolvida para uma situação específica voltada para empresas em um duopólio (FELICIANO, 2007).

Chegando já ao século XX, as pesquisas nessa área são mais frequentes e é dada uma maior importância por parte da comunidade, que não era recebida até o momento. Em 1913, Ernst Zermelo, um grande matemático e filósofo alemão, publicou em seu artigo *Uber eine Anwendung der Mengenlehre auf die Theorie des Schachspiels,* o teorema que mais tarde ficaria conhecido como classificação de um jogo determinado, utilizando como exemplo o jogo de xadrez, em que cada jogador teria uma estratégia vitoriosa, independente da do seu adversário (FELICIANO, 2007; BORTOLOSSI; GARBUGIO; SARTINI, 2017).

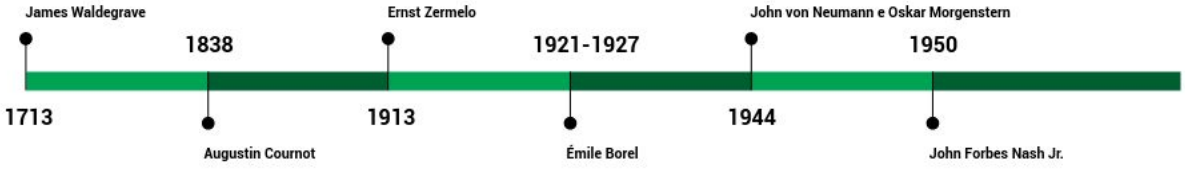
Outro grande matemático que deixou sua contribuição para a Teoria dos Jogos nessa época foi o francês Félix Édouard Justin Émile Borel, que publicou entre os anos de 1921 e 1927, diversos artigos sobre jogos de estratégia (CÂMARA, 2011). Inclusive propondo o jogo do Coronel Blotto, que foi implementado neste trabalho.

De acordo com Feliciano (2007), von Neumann demonstra que um jogo de soma zero com dois jogadores e diversas estratégias puras pode ser solucionado utilizando estratégias mistas, onde também é apresentada a representação de um jogo na sua forma extensa, discutida logo mais. Chegamos ao lançamento do livro de Neumann e Morgenstern em 1944, trazendo problemas econômicos para a Teoria dos Jogos, chamando muita atenção e tornando o seu estudo mais comum na época.

Nash (1950) citado por Bortolossi, Garbugio e Sartini (2017), apresenta o equilíbrio, que leva seu nome, em jogos não-cooperativos, provando isso com um jogo, na sua tese de doutorado realizado na Universidade de Princeton. Ele lançou quatro importantes artigos sobre o assunto, abordando desde o equilíbrio de Nash, jogos não-cooperativos, jogos cooperativos até a teoria da barganha, proposta por ele, em que Nash prova que existe uma

solução para esse problema em jogos cooperativos. O estudo de jogos não-cooperativos rendeu a Nash o prêmio Nobel de Economia em 1994, juntamente com John Harsanyi e Reinhard Selten. A Figura 1 apresenta uma síntese da história descrita.

Figura 1 - Linha do tempo



Fonte: Elaborada pelo autor.

Desse modo, percebe-se que a Teoria dos Jogos tem aplicação em diversas áreas da atuação humana, como foi mostrado a partir de sua evolução ao longo da história, com colaborações de vários ramos da ciência.

A escolha de decisões ótimas pode ser aplicada nos mais distintos campos de estudo, para isso ela se apoia em alguns conceitos fundamentais levando em consideração a lógica humana, dentre eles se encontra a Teoria da Decisão.

Anand (1995) que é citado por Paiva (2012), define a Teoria da Decisão como a melhor escolha, respeitando as crenças e desejos que um agente possui. Jianya (2016, p. 09) afirma que "a Teoria dos Jogos difere-se da Teoria da Decisão porque pelo menos duas entidades decisoras participam do processo".

Outros autores também definem a teoria da decisão como sendo a decisão do que escolher quando não se sabe o que acontecerá a partir dessa escolha, definido por (Campello de Souza, 2007) também mencionado por Paiva (2012), mostrando assim ser um dos pilares para a definição da Teoria dos Jogos, visto que tudo é motivado pela decisão do jogador, com o intuito de se alcançar algum objetivo. Para isso é necessário entender de qual forma é empregada esta teoria, logo, nada melhor do que iniciar definindo o objeto principal de estudo, o jogo.

2.1.1 O Jogo

Um jogo refere-se a uma situação de conflito ou concorrência na qual um ou mais jogadores escolhem ações considerando suas estratégias, buscando assim no final maximizar

um *payoff*. Contudo nem toda interação entre os jogadores acontece de forma concorrente. Em muitos casos se desenvolve a cooperação entre os mesmos (LANZOTTI, 2006; ROY et al., 2010).

Para se ter uma boa experiência e regras bem determinadas um jogo pode ser representado como um modelo formal, evitando que uma modelagem errada influencie nas escolhas tomadas (FELICIANO, 2007).

2.2.1 Elementos de um jogo

- Jogadores

O principal elemento que faz parte do jogo sem dúvida é o jogador, responsável pela tomada de decisões durante o jogo. Segundo Pereira (2006), os jogadores são os participantes que determinam quais ações serão adotadas no decorrer de um jogo. São eles que determinam as decisões. Cada jogador tem como objetivo maximizar o resultado com a escolha de suas ações, não se caracterizando apenas como pessoa, mas como qualquer entidade que pode influenciar no resultado do jogo (JIANYA, 2016).

- *Payoff*

Como foi citado na definição de jogo, o pagamento (*payoff*) é visto na maioria das vezes como sendo o principal objetivo quando se participa de um jogo. Sendo assim, pode-se definir *payoff* como sendo a recompensa ou pagamento obtido ao final do jogo, geralmente indicado de forma numérica associada a uma estratégia escolhida pelo jogador. Nem sempre é positivo, podendo variar desde uma perda até um ganho (PEREIRA, 2006; UCHÔA, 2015).

- Função de utilidade

O *payoff*, definido anteriormente, é decorrente de uma função a qual é dada o nome de função de utilidade, entendida como sendo uma equação que modela e mapeia os estados de um jogo para números reais (*payoff*), determinado para cada estratégia escolhida. Pode-se utilizar uma função de utilidade quando o jogador não sabe ao certo em qual estado se encontra (CARVALHO, 2013, UCHÔA, 2015).

2.1.2 Estratégias

Poundstone (1993) citado por Azevedo (2003, p. 184), afirma que: "estratégia, na Teoria dos Jogos, deve ser entendida como o conjunto de opções de ação que os jogadores têm para chegar a todos os resultados possíveis."

Estratégia seria então um outro grande ponto utilizado pela Teoria dos Jogos. Quando se abre e dispõe de uma gama de opções para os jogadores, pode-se implantar a Teoria dos Jogos para estudar qual a melhor estratégia a se escolher em uma determinada situação em que o jogador se encontrar.

Em um jogo existem algumas classificações para as estratégias, a depender de como elas se originam, existindo também o conceito de estratégia dominante, que está incluso dentro das outras classificações. Elas serão definidas com mais detalhes a seguir.

- Estratégia Pura

Jogos de estratégia pura são aqueles nos quais os jogadores não baseiam suas estratégias em aleatoriedade (AZEVEDO, 2003). Desse modo, é feita uma escolha "natural" dentre as opções disponíveis para o jogador.

A ação a ser realizada é determinada para cada situação em que o jogo se encontre, juntamente com a função de utilidade estabelecendo qual o *payoff* para aquela determinada ação, levando-se a uma possível determinação de qual estratégia será utilizada pelo adversário (FELICIANO, 2007; ARVELOS, 2009).

- Estratégia Mista

Bem intuitivo, neste tipo de jogo o agente não adota uma estratégia pura, como foi descrito anteriormente, mas decide usar uma distribuição de probabilidade sobre o perfil de estratégias que minimizarão o risco. O tomador de decisão usa uma loteria para definir a estratégia que deve ser adotada (PAIVA, 2012).

Desse modo, o jogador deve dispor de uma distribuição de probabilidade entre as estratégias disponíveis para ele, ajudando na sua escolha ao longo do jogo. A estratégia pura pode ser considerada um tipo específico de estratégia mista, em que a probabilidade é 1 (JIANYA, 2016).

- Estratégia Dominante

Segundo Jianya (2016, p. 12), “na Teoria dos Jogos, uma estratégia dominante é caracterizada por ser a melhor para um dos jogadores independentemente das estratégias utilizadas pelos demais”. Por esse motivo, entre outros, muitas vezes se torna fácil determinar a estratégia dominante de um jogo, que é uma das primeiras a ser enxergadas pelos jogadores.

“Uma estratégia dominante de um jogador se caracteriza por produzir o maior ganho possível dentro das estratégias disponíveis para todas as ações possíveis dos outros jogadores” (JIANYA, 2016, p. 12), mostra a importância e a implicação que se tem ao escolher uma determinada estratégia e como a estratégia dominante se destaca das demais disponíveis.

De acordo com Feliciano (2007), não existe uma vantagem de se escolher uma estratégia secundária em detrimento de uma estratégia dominante. A estratégia dominante sempre garantirá o melhor resultado para o jogador, independente das escolhas do oponente.

Ainda segundo Jianya (2016), é possível definir que há um equilíbrio de estratégias mistas quando nenhum dos jogadores tem uma estratégia que ofereça um maior ganho em relação às demais.

- Equilíbrio de Nash

Discutindo o assunto que trata sobre equilíbrio nos jogos, levando em consideração as estratégias disponíveis para os jogadores, área que concentrou parte dos estudos de John Nash, ele definiu o ponto de equilíbrio em jogos não-cooperativos, registrados em artigos publicados por ele próprio, formando sua tese de doutorado (BORTOLOSSI; GARBUGIO; SARTINI, 2017).

Conceito esse que ficou intitulado como ponto de equilíbrio e hoje também é conhecido como Equilíbrio de Nash, definido por Jianya (2016, p. 13), “o Equilíbrio de Nash configura uma situação onde cada jogador não possui motivação em alterar sua estratégia dentro do jogo se os outros jogadores também assim não o fizerem”.

Em um jogo equilibrado é possível afirmar que todos os jogadores escolheram atualmente suas melhores estratégias em resposta às estratégias de seus adversários (FIANI, 2004 apud CÂMARA, 2011).

2.1.3 Formas de representação

Para representar os jogos de modo a facilitar o entendimento foram desenvolvidas algumas formas de representação, a forma normal ou estratégica e a forma extensiva, cada uma com as suas peculiaridades.

- Forma Normal ou Estratégica

A forma de representação normal é exibida em forma de tabela, com os jogadores dispostos em linhas e colunas, onde essa tabela é preenchida com as estratégias dos respectivos jogadores, seguindo a organização de linha e coluna para apresentar os devidos *payoffs* por cada estratégia adotada. É utilizada principalmente para representação de jogos classificados como de decisão simultânea, que são basicamente jogos onde os jogadores escolhem suas estratégias concomitantemente (SARTINI et al., 2004; FIANI, 2009; PAIVA, 2012). Pode tornar-se de difícil legibilidade quando aplicada para um número superior a dois jogadores, deixando de ser bidimensional (FELICIANO, 2007). Um exemplo de representação na forma normal é mostrado no Quadro 1.

Quadro 1 - Jogo na forma normal

	Jogador 2	
Jogador 1	**L**	**R**
T	2, 2	4, 0
B	1, 0	3, 1

Fonte: Myerson (1991, com adaptações).

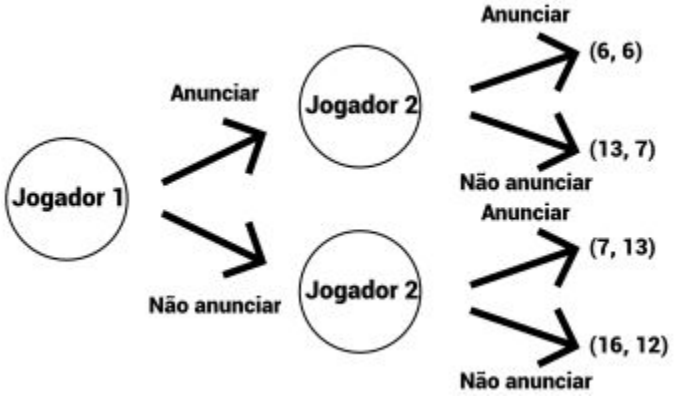
T e B são as estratégias disponíveis para o Jogador 1, já L e R as estratégias do Jogador 2; o primeiro elemento do par ordenado é o *payoff* do Jogador 1 e o segundo elemento é o *payoff* do Jogador 2. Ao interceptar o resultado das estratégias escolhidas obtêm-se o *payoff* resultante para cada um.

- Forma Extensiva

Do outro lado se tem a forma de representação extensiva que é baseada na teoria dos grafos, tendo sido apresentada pela primeira vez no livro *The Theory of Games and Economic Behaviour* (FELICIANO, 2007).

Representando em forma de árvore todas as estratégias disponíveis para o jogo, essa estruturação tem uma melhor correspondência para demonstrar jogos do tipo sequencial, exatamente o oposto de jogos de escolha simultânea em que os nós seriam uma etapa/estado do jogo e as arestas seriam as decisões disponíveis para o jogador da vez. O jogador vai determinar a escolha de sua jogada a partir do estado atual, afetando diretamente a próxima jogada do seu adversário (SARTINI et al., 2004; FIANI, 2009; PAIVA, 2012; JIANYA, 2016).

Figura 2 - Jogo na forma extensiva



Fonte: Jianya (2016, com adaptações).

No exemplo da Figura 2 em que Jianya traz a especificação de um jogo envolvendo a tática de propaganda entre duas empresas, é possível se entender a ideia de representação na forma extensiva e a sua identificação com jogos sequenciais. Os *payoffs* ficam localizados nas folhas da árvore, enquanto a raiz representa o Jogador 1, escolhendo o ramo que será seguido dependendo da sua escolha de estratégia.

Essa forma de representação traz mais forte a ideia de estratégia pura entre o conjunto de estratégias dos jogadores, com a organização das ações que podem e quando podem ser realizadas, introduzindo a ideia de mudança de estratégia, sendo influenciada pela jogada anterior do adversário. É necessária uma descrição da ordem dos movimentos de quando os jogadores podem realizar determinadas ações (TADELIS, 2013).

2.1.4 Classificação dos jogos

A Teoria dos Jogos trabalha com uma grande quantidade de jogos. Ao longo do tempo e com o avanço dos estudos foram se definindo classificações para agrupar esses jogos, facilitando assim o uso dessa teoria para associá-la com situações do dia a dia.

Existem várias características que podem servir como ponto para uma classificação, por exemplo, podem ser definidas tanto a questão da cooperatividade entre os jogadores no jogo, a dinamicidade desenvolvida, o nível de informação que se tem, seja ela completa ou prévia sobre, de acordo com o modo como se transcorre o jogo, quanto a simetria entre as jogadas. Também pode-se classificar relacionado a como é distribuído o ganho entre os jogadores e por fim baseado na determinação de estratégias disponíveis para aquele jogo.

No decorrer desta Seção são apresentadas cada uma das formas de classificação dos jogos dentro da área de Teoria dos Jogos. Juntamente com as definições estarão presentes exemplos de jogos que são classificados, buscando ajudar o leitor a compreender melhor a definição de cada classificação. Os jogos utilizados como exemplos são descritos em mais detalhes no Apêndice A - Jogos Exemplos.

- Jogos cooperativos e não-cooperativos

Em um jogo cooperativo os jogadores adotam determinado curso de ação que reservam boas recompensas a todos os envolvidos, aumentando o *payoff* dele e dos demais jogadores que contribuem entre si no grupo. Já nos jogos não-cooperativos existe uma combinação de estratégias que oferece os ganhos mais atraentes para um, mesmo que à custa de perdas para os demais, maximizando com isso o seu *payoff* (BÊRNI, 2004; UCHÔA, 2015). Essa é uma das classificações mais importantes na Teoria dos Jogos. Assim, pela sua representatividade, de acordo com Tadelis (2013, p. 204, tradução do autor):

> Enquanto um jogador mantiver sua reputação de cooperar, outros jogadores confiarão nele e responderão da mesma forma. Se um jogador deixar de cooperar em qualquer estágio, ele perderá sua boa vontade e os jogadores passarão para uma fase não cooperativa de seu planejamento.

Como exemplo de um jogo cooperativo temos o jogo Batalha dos Sexos, visto que para uma maior satisfação dos jogadores é necessário que exista uma cooperação entre eles, aceitando e ajudando na decisão do outro.

Já para jogos não-cooperativos temos o clássico Jogo pedra-papel-tesoura, em que os jogadores não têm a mínima intenção de cooperação com o adversário para conseguir algum objetivo; o intuito é justamente o contrário, que o adversário perca para que ele ganhe, sempre buscando diminuir o ganho do adversário.

- Jogos estáticos e dinâmicos

Essa classificação é referente ao número de jogadas (interações) realizadas pelos jogadores, ao número de vezes que as estratégias são confrontadas, que de acordo com Bêrni (2004, p. 21):

> Os jogos estáticos são aqueles em que existe apenas uma ocasião de interação estratégica entre os jogadores, não necessariamente simultânea. Quando os jogadores se defrontam mais de uma vez com o problema da escolha associada à ação do outro jogador, diz-se que eles estão envolvidos em um jogo dinâmico ou superjogo. Neste caso, há mais de uma interação entre os agentes.

O Jogo pedra-papel-tesoura também é um exemplo de jogo estático, levando em consideração a interação entre os jogadores, que expressam suas escolhas apenas uma vez a cada partida, decidindo assim quem foi o ganhador.

O tão conhecido Jogo da Velha pode ser utilizado como exemplo para um jogo dinâmico já que uma partida se desenrola com várias interações feitas pelos jogadores, iniciando com o tabuleiro vazio até preenchê-lo totalmente ou até que algum jogador chegue à vitória.

- Jogos de informação perfeita e imperfeita

Classificam-se dessa forma os jogos de acordo com as informações que os jogadores têm anteriormente ao início do jogo. Segundo Bêrni (2004) e Kelly (2003) citadas por Silveira (2008), no jogo de informação perfeita cada jogador sabe tudo ou boa parte do que ocorreu antes do início do jogo, incluindo também o que o outro jogador escolheu até o momento, ajudando de certa forma a ele escolher qual estratégia adotar a partir daquele momento. Já em um jogo de informação imperfeita o jogador não tem informações prévias sobre a antecedência do jogo, tentando assim antecipar as ações do seu adversário.

O Jogo do Ultimato pode ser considerado um jogo de informação perfeita, já que o jogador tem conhecimento da estratégia adotada pelo seu adversário até o momento, como também todas as outras informações anteriores ao jogo.

Do outro lado, o Jogo pedra-papel-tesoura é tido como um jogo de informação imperfeita, visto ser um jogo estático, em que o jogador não tem nenhuma informação anterior à partida que possa lhe ajudar a prever alguma estratégia que seja para tirar proveito disso, agindo dessa forma na tentativa de adivinhar o que seu oponente vai escolher.

- Jogos de decisão simultânea e sequencial

É uma classificação bem intuitiva como o nome sugere, mas que tem grande importância para alguns conceitos da Teoria dos Jogos.

Em um jogo de decisão simultânea os jogadores expressam suas escolhas ao mesmo tempo um do outro, sem a possibilidade de observar a decisão do adversário. Já em jogos de decisão sequencial existe uma ordem para a tomada de ações, em que um jogador escolhe uma ação como resposta à ação do seu adversário (BÊRNI, 2004; RIBEIRO et al., 2013).

Um jogo dinâmico pode ter decisões sequenciais ao ponto que um jogo estático pode ter decisões simultâneas (BÊRNI, 2004). Isso pode levar a uma confusão entre as classificações, diferindo-se pelo fato de que simultânea/sequencial se refere à ordem de escolha, enquanto estática/dinâmica diz respeito à quantidade de escolhas.

Cara ou Coroa é um jogo que se classifica como de decisão simultânea já que os dois jogadores escolhem suas opções ao mesmo tempo, não há uma sequência entre a decisão de um jogador e a do seu adversário.

O Jogo da Velha é um jogo sequencial, em que os jogadores vão intercalando suas jogadas, seguindo uma sequência, uma jogada após a do adversário até que o tabuleiro seja completamente preenchido ou um dos dois consiga a vitória.

- Jogos simétricos e assimétricos

Uma classificação tem relação quanto à escolha das estratégias e os seus respectivos *payoffs*. De acordo com Bêrni (2004), em um jogo simétrico, a adoção de determinadas estratégias por parte dos jogadores gera resultados que se as ações forem invertidas, os ganhos são invertidos também, ou seja, os *payoffs* se mantêm correspondentes, independente do lado. Os jogos assimétricos modelam justamente o contrário.

Um exemplo de jogo simétrico seria o Jogo pedra-papel-tesoura, em que quando se invertem as estratégias utilizadas os pagamentos pelas estratégias também são invertidos, de forma uniforme. Um jogador escolhendo pedra e o adversário escolhendo tesoura, o primeiro será vencedor, ocorrendo o contrário se essas escolhas forem invertidas, quando o segundo sairia vencedor.

O Jogo do Ultimato é considerado um jogo assimétrico porque não é possível garantir que serão utilizadas as mesmas estratégias entre os jogadores, já que são dois tipos de jogadores diferentes e o objetivo final das estratégias deles não é o mesmo, cada qual com a sua peculiaridade.

- Jogos de informação completa e incompleta

Nos jogos de informação completa, os jogadores conhecem as regras do jogo e os pagamentos a serem recebidos por cada jogador, dependendo da combinação de estratégias adotadas pelos jogadores, o que influencia diretamente nas escolhas. Já em jogos de informação incompleta, pelo menos uma das funções que define o pagamento de um jogador é desconhecida para os outros (PAIVA, 2012; RIBEIRO et al., 2013).

Jogo da Velha é um jogo de informação completa, em que jogadores têm conhecimento das estratégias disponíveis juntamente com os pagamentos por cada uma das estratégias; eles também têm conhecimento das jogadas efetuadas pelo seu adversário no decorrer da partida.

Como jogo de informações incompletas pode-se citar o Jogo dos bens-públicos, em que os jogadores não possuem informações sobre todas as opções referentes ao jogo, como por exemplo, o fator pelo qual serão multiplicados os investimentos dos jogadores.

- Jogos de soma zero e soma não zero

Outra classificação que tem grande importância na Teoria dos Jogos é a classificação de jogos de soma zero, principal área de análise dos estudos de John von Neumann. O autor propõe que jogos de soma zero podem ser resolvidos utilizando cálculos matemáticos. Posteriormente em sua obra *The Theory of Games and Economic Behavior,* de 1944, é realizado o estudo mais aprofundado de jogos de soma zero, juntamente com Oskar Morgenstern (FIANI, 2006).

Segundo Bêrni (2004) e Paiva (2012), um jogo de soma zero é representado quando a recompensa de um jogador é igual à perda do adversário, em que ele tenta sempre reduzir o ganho do oponente para maximizar o seu. Como o próprio nome sugere, a soma das vitórias com as derrotas de todos os jogadores é igual a zero.

Já em um jogo em que todos os jogadores ganham ou perdem simultaneamente, se classifica como um jogo de soma não zero, quando a soma das vitórias e derrotas entre os jogadores não é nula.

O Cara ou Coroa é um exemplo de jogo de soma zero, em que para que o jogador ganhe, o seu oponente deve perder. Como é um jogo sem pontuação, a vitória de um jogador é diretamente proporcional à derrota de outro. Em jogos com pontuação funciona da mesma maneira, a pontuação do vencedor deve ser inversamente proporcional à pontuação do seu adversário, para que ele ganhe.

Batalha dos Sexos é classificado como um jogo de soma não-zero porque os jogadores podem chegar a um acordo e no fim todos saírem satisfeitos, sem que para isso seja escolhida a estratégia de apenas um dos dois.

- Jogos determinados e indeterminados

Segundo Feliciano (2007) citado por Arvelos (2009), caso um jogo ofereça a possibilidade de no mínimo um empate sempre, desde que o jogador siga um roteiro de ações desde o início do jogo e evite dessa forma a derrota, ele se classifica como um jogo determinado. Caso o jogo seja imprevisível e o jogador por mais racional que seja não consiga sempre uma tática que lhe assegure evitar a derrota e não exista uma forma de deduzir isso, este se classifica como um jogo indeterminado.

O Jogo da Velha é um exemplo básico de jogo determinado, existe pelo menos uma determinada estratégia em que o jogador pode garantir sempre, no mínimo um empate, independente da estratégia adotada pelo seu adversário.

Como exemplo de jogo indeterminado pode ser usado o Jogo pedra-papel-tesoura, no qual não existe uma estratégia que assegure ao jogador o empate, mesmo tendo informações adicionais. Toda estratégia é passível de derrota, pois é um jogo imprevisível.

Após a apresentação das formas de classificação acima, pode-se induzir algumas restrições para a classificação de jogos, esta que não é compatível uma com as outras, como

por exemplo, a de que um jogo determinado não pode ser simultâneo, como provado pela própria definição.

Concluída a apresentação de alguns conceitos básicos da Teoria dos Jogos, entre eles os que serão utilizados no jogo Coronel Blotto, na sua parte conceitual e contextual, agora será debatido um pouco dos assuntos que dizem respeito à parte mais funcional, de construção do jogo e aspectos adotados para a implementação do mesmo.

2.2 Paradigma de programação funcional

Como já foi antecipado no objetivo deste trabalho, antes de comentar sobre a linguagem de programação Haskell, debateremos um pouco sobre o paradigma de programação ao qual ela se classifica.

Paradigma é uma ideia ou modelo a se seguir, levando para o campo da computação e programação em particular, segundo Van Roy e Haridi (2004) e Van Roy (2009) citados por Xavier (2014), seria um sistema formal que vai orientar o programador como deve prosseguir na construção do sistema, fornecendo conjuntamente uma gama de técnicas para programar e organizar os pensamentos frente ao problema ao qual se propõe a resolver.

Cabe ao programador escolher qual paradigma de programação irá utilizar para determinado projeto o qual ele tem mais afinidade. O paradigma vai definir como o problema será representado, como também vai influenciar em todo o processo de desenvolvimento, por isso deve ser bem escolhido (SOUSA; DIAS JR; FORMIGA, 2014).

Existem quatro tipos principais de paradigmas de programação, sendo eles: o paradigma imperativo, orientado a objetos, lógico e o paradigma funcional. Esses paradigmas possuem as mais diversas aplicações e cabe ao programador escolher qual o mais indicado para o seu problema.

A linguagem que foi escolhida para ser debatida neste trabalho é uma linguagem que segue as diretrizes do paradigma funcional, por isso serão trabalhados mais detalhes a respeito desse paradigma. São apresentados uma visão geral do mesmo, algumas das suas principais características, a motivação para sua escolha, seus pontos positivos em relação aos demais paradigmas, como também mais à frente é apresentada uma das principais linguagens que seguem esse paradigma, esclarecendo melhor alguns pontos da programação funcional.

Como o próprio nome é bem sugestivo e direto, o paradigma de programação funcional tem sua modelagem fundamentada em agrupamentos de funções matemáticas, em

que com esse relacionamento pretende-se chegar ao resultado pretendido. No geral, assim como na Matemática, existe um espaço de entrada ou domínio e um espaço de resultado ou faixa conhecido na matemática como imagem (TUCKER; NOONAN, 2010).

De acordo com Hughes (1989), Van Roy e Harefidi (2004), Scott (2008), Brookshear (2012) e Simão et al. (2012a) citados por Xavier (2014, p. 15):

> Um programa é construído como uma função principal, que recebe argumentos de entrada (e.g. funções) e resulta em uma saída. A função principal é tipicamente definida em termos de outras funções, que também são definidas em termos de outras funções, dessa forma até que as funções elementares sejam as primitivas da linguagem de programação. Recursão é uma característica elementar (e recorrente) no paradigma funcional.

Expressões podem ser vistas como um agrupamento maior nesse paradigma, elas podem conter os próprios valores, "variáveis", funções e também outras expressões. As funções, como já definidas, podem receber valores na entrada, em que aplicando-se obtém-se as saídas das mesmas, ao final reduzindo e avaliando as expressões (ALLEN; MORONUKI, 2016). Esses são termos bastante utilizados em linguagens funcionais (que implementam o paradigma funcional).

O paradigma de programação funcional tem uma enorme importância desde o seu surgimento, início da década de 60. É um dos principais paradigmas de programação que não seguem a forma imperativa, criado pela necessidade dos pesquisadores da área da inteligência artificial (IA) e de seus campos de estudo, áreas que com o avanço da IA não estavam tendo suas necessidades atendidas pelos paradigmas de programação existentes naquela época (TUCKER; NOONAN, 2010).

Seu lançamento foi marcado pelo anúncio de apresentação da linguagem LISP, criada em 1958 por John McCarthy, após estudo das necessidades que haviam na época para facilitar a experiência com sistemas que estavam sendo desenvolvidos pelo grupo do MIT. Ele estabeleceu que era necessário o desenvolvimento de uma nova linguagem, apoiando-se em um formalismo particular entre funções recursivas e expressões simbólicas (MCCARTHY, 1960).

McCarthy et al. (1965) citado por Tucker E Noonan (2010) afirma que no ano de 1965 foi lançado o manual da linguagem, *LISP 1.5 Programmer's Manual* com apenas 106 páginas, um número impressionante para a descrição de uma linguagem de programação. Exemplo do tão conhecido cálculo fatorial na linguagem LISP:

```
(defun fatorial (n)
  (if (= n 0)
      1
      (* n (fatorial (- n 1)))))
```

De acordo com McCarthy et al. (1965 apud TUCKER; NOONAN, 2010, p. 362), a linguagem LISP é descrita da seguinte forma:

> A linguagem Lisp serve primariamente para processamento simbólico de dados. Ela tem sido usada para cálculos simbólicos em cálculo diferencial e integral, projeto de circuitos elétricos, lógica matemática, jogos e outros campos da inteligência artificial.

Antes mesmo disso, em 1930, Alonzo Church desenvolveu um sistema formal, chamado lambda cálculo, juntamente com a noção de computabilidade e outros conceitos fundamentais para o paradigma funcional (BARENDREGT; BARENDSEN, 1988).

De acordo com Tucker e Noonan (2010), o cálculo lambda é base para a programação funcional, que expressa muito resumidamente a definição de funções sem nomes, com a utilização de parâmetros, manipulados para gerar um resultado, assim como na Matemática.

Após um breve histórico e apresentação resumida desse paradigma de programação são debatidos alguns pontos que se destacam do paradigma funcional em relação aos demais, que o tornam especial e altamente poderoso, como por exemplo, área de IA, com multiprocessamento sem muito esforço.

Primeiramente, é necessário fazer uma distinção que envolve a classificação de linguagens funcionais, que podem ser puras ou impuras. A pureza de uma linguagem diz respeito a um termo conhecido como transparência referencial, definido em Allen e Moronuki (2016) como sendo uma função que retorna o mesmo resultado, sempre quando avaliada com os mesmos parâmetros, independentemente da situação, assim como na Matemática. Haskell é um exemplo de linguagem puramente funcional, diferentemente da linguagem funcional LISP, apresentada anteriormente, que apresenta características do paradigma imperativo que a torna uma linguagem impura.

A ideia de variável, citada anteriormente, não segue a mesma linha de variável das linguagens de programação imperativas, ligadas a endereços de memória ou rotinas de atribuição. Meira (1988) citado por Schmitt (2003, p. 39) diz que:

> Usando apenas definição de funções, como na matemática, e aplicação destas funções a argumentos, como mecanismos, o paradigma funcional exclui atribuição e controle como elementos de programação.

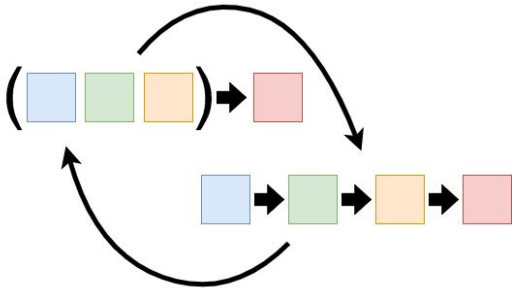
Utiliza composição de funções, ou seja, funções são passadas como parâmetros para outras funções, em que essas funções também retornam funções, conhecido também como uma forma funcional, deixando claro o termo funcional desse paradigma (TUCKER; NOONAN, 2010), levando a classificar funções como sendo de primeira ordem (SCHMITT, 2003). Vale lembrar que podem existir funções sem argumento, apenas retornando outra função.

De acordo com Sebesta (2011), a definição de uma linguagem funcional fornece um conjunto de funções da própria linguagem, como também algumas estruturas para representar os dados, que serão usados como parâmetros, valores de retorno, etc., assim como em outros paradigmas. A partir disso, o programador pode incrementar e criar suas próprias funções.

Funções podem ou não ser nomeadas, utilizando para isso princípios do cálculo lambda, ficando a cargo do programador no momento da construção, a depender da necessidade. Essas funções, assim como na Matemática, podem ser manipuladas utilizando as leis da álgebra. Como esses conceitos são muito próximos da Matemática isso faz com que o paradigma funcional seja até melhor compreendido por quem tem um conhecimento básico na área, não exigindo tanto como os demais na área da Programação (SCHMITT, 2003).

Um recurso muito interessante que linguagens funcionais podem oferecer é chamado de *curry*. Ele pode ser determinado como sendo uma "transformação" de uma função com muitos argumentos para uma composição de várias funções de apenas um argumento (SCHMITT, 2003).

Figura 3 - Processo de currificação de funções



Fonte: (JUNKER, 2017).[1]

[1] Disponível em: https://link.medium.com/hLft1PNuj1. Acesso em: 30 out. 2019.

Como mostrado na Figura 3, uma função com três argumentos poderia ser transformada em três funções unárias, onde a função recebe um argumento e retorna uma função unária, até chegar ao resultado esperado.

Isso permite a aplicação parcial de funções, que pode ser muito proveitoso em algumas situações, como por exemplo, tornar a composição de funções mais flexível, o que implica também na simplificação do código, facilitando seu entendimento e reduzindo consideravelmente sua complexidade (JUNKER, 2017).

Em linguagens puramente funcionais não existem operadores para atribuição de valores, como existem em linguagens orientadas por objeto, por exemplo, o que leva a não se ter uma ideia de endereço de memória, um princípio do paradigma funcional (SEBESTA, 2011), levando a outro princípio que seria o conceito de estado do programa, que não existe nesse paradigma (TUCKER; NOONAN, 2010). O estado de uma aplicação seria uma coleção de chave e valor, em que a chave é uma variável que aponta para um determinado valor que está armazenado na memória (SEBESTA, 2011).

A recursividade, característica importante da computação, para resolução de grandes problemas, é presente na programação funcional, Allen e Moronuki (2006, p. 270, tradução do autor) definem recursão como sendo:

> A recursão está definindo uma função em termos de si mesma, por meio de expressões de autorreferência. Isso significa que a função continuará a se chamar e repetir seu comportamento até que alguma condição seja atendida para retornar um resultado.

Na programação funcional não existe a ideia de estruturas de repetição (por exemplo, *for* e *while*) para realizar iterações, como em linguagens imperativas, já que elas são validadas por variáveis de controle. Para suprir essa necessidade é realizada justamente a implantação da recursividade. A utilização de recursividade pode levantar alguns entraves, como uma maior demanda e consequentemente um maior tempo de processamento em relação à resolução iterativa. Isso pode ser amenizado com a adoção de alguns métodos como a recursão em cauda, por exemplo, fazendo com que a chamada recursiva seja a última a ser executada (SEBESTA, 2011).

2.2.1 Paradigma Funcional x Paradigma Orientado a Objetos

Essa Seção reserva uma breve comparação entre os paradigmas funcional e orientado a objetos, levantando algumas vantagens e desvantagens na utilização dos mesmos.

O paradigma orientado a objetos tem seus primeiros traços na história registrados com o lançamento da linguagem SIMULA 67 na década de 60, tendo maior conhecimento nos anos 80, com o lançamento do Smalltalk 80, considerada por muitos como a primeira linguagem orientada a objetos pura (SEBESTA, 2011). Hoje é vasto o número de linguagens que seguem e adotam esse paradigma, como Python[2], Java[3], C++[4], entre outras.

Nesse paradigma o mundo no qual o problema está contido é mapeado para objetos, incluindo os seus atributos e suas formas de interação com os demais objetos (CALÔNEGO JUNIOR, 1997). Essa abstração torna muitas vezes mais clara a visualização do problema por parte da equipe de desenvolvimento.

A principal diferença entre os dois paradigmas pode ser considerada a existência de um estado na programação orientada a objetos (POO), com sua ausência na programação funcional (WARBURTON, 2016).

Uma linguagem funcional pode trazer um código sintático mais limpo, favorecendo um maior entendimento da lógica central da aplicação. Pode ser considerada também a utilização de recursão em oposição à iteração, trazendo muitas vezes maior clareza. A POO pode gerar ações colaterais, o que não é encontrado em linguagens puramente funcionais graças à transparência referencial, citada anteriormente (SEBESTA, 2011).

Segundo Sebesta (2011), em relação à velocidade de execução, a programação funcional perde um pouco ainda por seguir princípios interpretados, em que a POO segue a rotina de compilação. Com o avanço estão sendo lançados um grande número de compiladores para linguagens funcionais, acarretando na diminuição dessa discrepância, mesmo não sendo tão importante em sistemas onde o tempo de resposta não é algo tão requisitado.

Hoje, o suporte a expressões lambda já é oferecido por algumas linguagens POO, como a versão 8 do Java, trazendo os pontos positivos da programação funcional, como a parametrização no comportamento do código (WARBURTON, 2016), mostrando como os

[2] www.python.org
[3] www.java.com
[4] www.cplusplus.org

paradigmas “aprendem” uns com os outros. É possível realizar uma comparação em diversos pontos, como é mostrado no Quadro 2.

Quadro 2 - Comparativo PF vs POO

	Programação funcional	**Programação orientada a objetos (POO)**
Definição	A programação funcional enfatiza uma avaliação de funções.	POO é baseada no conceito de objetos.
Dados	A programação funcional usa dados imutáveis.	POO usa dados mutáveis.
Modelo	A programação funcional segue um modelo de programação declarativa.	POO segue um modelo de programação imperativa.
Execução	Na programação funcional, as instruções podem ser executadas em qualquer ordem.	Na POO as instruções devem ser executadas em ordem específica.
Iteração	Na programação funcional, a recursão é usada para iterar sobre dados.	Na POO os loops são usados para iterar sobre dados.
Elementos	Os elementos básicos da programação funcional são funções.	Os elementos básicos da programação orientada a objetos são objetos e métodos.

Fonte: (EDUCBA[5], com adaptações)

Os códigos a seguir mostram um simples comparativo para o cálculo da soma da série de *fibonacci*, nas linguagens Haskell (funcional) e Java (POO), respectivamente:

```
fib 0 = 0
fib 1 = 1
fib n = fib (n-1) + fib (n-2)
```

```
public long fibo(int n) {
    if (n < 2) {
        return n;
    } else {
        return fibo(n - 1) + fibo(n - 2);
    }
}
```

Apresentados os conceitos sobre o paradigma funcional, o mesmo foi escolhido para demonstrar o poder que ele pode oferecer, onde não se restringe apenas à área de IA, a qual foi proposto seu desenvolvimento inicialmente, mas como um propósito mais geral, que pode

[5] Disponível em: www.educba.com/functional-programming-vs-oop. Acesso em 02 nov. 2019.

ser utilizado em diversas áreas, desmistificando algumas crenças que existem acerca do paradigma funcional e a produtividade que o próprio pode oferecer.

A título de conhecimento, a empresa Nubank utiliza programação funcional desde a sua criação, tida por ela como o melhor paradigma para os desafios que eles enfrentam e o que ajuda no reaproveitamento de recursos. Eles se aproveitam principalmente das seguintes vantagens: um código mais curto e objetivo, a imutabilidade, previsibilidade e disponibilidade para programação paralela, já que não existem efeitos colaterais (REDAÇÃO NUBANK, 2019).

E como se trata do mundo da programação, a citação de O'Sullivan, Goerzen e Stewart (2008, p. 71, tradução do autor) cai bem: "fazemos isso não porque as técnicas imperativas são ruins, mas porque em uma linguagem funcional outras técnicas funcionam melhor".

2.3 Haskell

A linguagem de programação que é apresentada neste trabalho é a linguagem Haskell[6]. São debatidas algumas características dela, assim como a motivação para a sua utilização.

Lipovača (2011) afirma que Haskell é uma linguagem puramente funcional. Isso pode ser confirmado pela existência da transparência referencial em seus princípios. Funções não emitem efeitos colaterais, o mesmo resultado é dado sempre para o mesmo parâmetro.

> Haskell é uma linguagem de programação avançada puramente funcional. Um produto de código aberto com mais de vinte anos de pesquisa de ponta, permite o rápido desenvolvimento de softwares corretos, concisos e robustos. Com forte suporte à integração com outras linguagens, simultaneidade e paralelismo integrados, *debuggers*, *profilers*, bibliotecas ricas e uma comunidade ativa, Haskell facilita a produção de software flexível, sustentável e de alta qualidade. (HASKELWIKI…, 2003, tradução do autor).

A história da linguagem Haskell se iniciou no ano de 1987, em uma conferência sobre linguagens de programação funcional e arquitetura de computadores, em Portland, Oregon, para tratar questões que haviam surgido entre a comunidade da programação funcional. Na conferência, ficou determinado que seria desenvolvida uma nova linguagem puramente funcional, comum, pelo comitê organizador (HUDAK et al., 2007).

[6] www.haskell.org

Antes mesmo disso, já havia um grande debate na comunidade em torno do tema sobre avaliação preguiçosa em linguagens de programação no final dos anos 70 e início dos anos 80. Tema esse que influenciou o desenvolvimento de várias linguagens na época (HUDAK et al., 2007). Schmitt (2003) afirma que:

> A linguagem Haskell foi lançada no fim dos anos 80 em uma tentativa de juntar muitas idéias na pesquisa de PF. É similar a ML[7] em razão de usar uma sintaxe semelhante; tem escopo estático, é fortemente tipificada e usa o mesmo método de inferência de tipos.

A linguagem Haskell teve uma grande relação em seu ponto de partida com a linguagem funcional Miranda[8], e recebeu o nome de Haskell como uma forma de homenagear o matemático estadunidense Haskell Curry após um longo debate entre o comitê (HUDAK et al., 2007), pela sua influência no cálculo lambda e também a técnica de *currying*, apresentada anteriormente como princípio de linguagens funcionais e que também rendeu homenagens ao seu nome (BARENDREGT; BARENDSEN, 1988).

Foram adotados alguns pontos para encorajar o desenvolvimento, como por exemplo, ser apropriada para ensino, pesquisa e desenvolvimento de grandes sistemas, apresentar uma sintaxe e semântica formais, estar disponível livremente, inclusive para edição e distribuição, basear-se num amplo acordo da equipe, dentre outros, o que de fato não ocorreu por completo (HUDAK et al., 2007).

Após diversas reuniões foi lançada uma primeira versão em 1º de abril de 1990. A partir daí ocorreram sucessivas melhorias em toda a estrutura da linguagem, chegando a uma versão estável em fevereiro de 1999. Em dezembro de 2002 foi publicado um livro pela Universidade de Cambridge com um relatório de 260 páginas (HUDAK et al., 2007).

2.3.1 Características

Como já foi possível abstrair algumas informações a respeito dessa linguagem, ela oferece um considerável número de características que a torna destacável entre as demais, agora serão elencadas algumas dessas características concedidas ao desenvolvedor.

Uma das suas principais características é que Haskell tem uma avaliação preguiçosa, um dos princípios para o seu desenvolvimento, como dito anteriormente. Isso faz com que ela

[7] Uma linguagem de programação funcional.
[8] miranda.org.uk

se torne altamente eficiente em diversos casos, executando avaliações apenas quando isso se fizer necessário (O'SULLIVAN; GOERZEN; STEWART, 2008). Muitas vezes não é interessante percorrer toda uma lista para se obter um resultado.

```
add x y = x + x
```

No exemplo acima o parâmetro **y** nunca é utilizado no corpo da função, então uma chamada a função do tipo ***add 5 (2/0)*** não quebraria a execução, mesmo o segundo argumento não sendo uma expressão válida. Haskell avalia apenas o que é realmente necessário.

Haskell oferece uma variedade, necessária, para representar tipos de dados (e. g. Int, Bool, Char) juntamente alinhada com a inferência de tipo, em que o programador não fica encarregado de declarar de qual tipo é uma variável, o próprio Haskell faz isso por ele. Também é muito interessante porque ajuda a identificar erros em tempo de compilação (LIPOVAČA, 2011).

Existe também a ideia de *Typeclass*, que consiste em agrupar tipos que têm características em comum. Através de uma interface, por exemplo *Eq*, tipos podem ser comparados pela igualdade.

A linguagem oferece a possibilidade de o programador criar os seus próprios tipos de dados, como por exemplo a definição de um tipo Bool, que representa um valor verdadeiro ou falso:

```
data Bool = False | True deriving (Ord)
```

Temos uma construção bastante simples, porém altamente legível. Definindo o tipo *Bool* com a palavra-chave *data* seguido dos tipos que ela pode assumir, *False* | (ou) *True*. O restante da sentença *deriving (Ord)* indica que esse tipo deriva de *Ord*. Com isso os valores podem ser comparados (LIPOVAČA, 2011).

De acordo com Lipovača (2011) e Allen e Moronuki (2016), a sintaxe da linguagem pode ser considerada bem rígida, desde o alinhamento até à definição de tipos de dados, como demonstrado acima, e isso tem seu ganho na organização do código fonte, favorecendo consequentemente a legibilidade, como também é recomendada a utilização de *camel case*.

Para armazenar conjuntos de valores se tem por padrão as listas, que guardam valores do mesmo tipo e de tamanho não mensurado, ou variável, juntamente com uma série de

funções para fazer a manipulação. Por outro lado, existem as tuplas, que são estruturas que podem conter valores de tipos diferentes, mas que tem um tamanho fixo, diferente das listas.

Um recurso interessante do Haskell que merece ser lembrado é o de compressão de listas, que funciona assim como na Matemática.

```
[ x * 2 | x <- [1..10]]
```

Isso vai gerar uma lista que tem como resultado o dobro de cada número de 01 a 10, deixando a sintaxe muito mais elegante e eficiente (LIPOVAČA, 2011). É possível criar uma lista de números "infinita" e Haskell só vai percorrer o que for necessário, graças à avaliação preguiçosa, como por exemplo, no trecho de código abaixo definindo a lista (linha 1) e então recuperando os 5 primeiros elementos (linha 2), a lista **a** apenas será processada neste instante e enquanto for realmente necessário, neste caso até o quinto elemento seja encontrado.

```
1. a = [5..]
2. take 5 a
```

Para melhorar o design do código o comitê optou por uma escrita seguindo uma certa indentação, especialmente na implementação de funções. Funções podem ser aplicadas de modo infixado (linha 1), onde o nome da função fica entre os argumentos da função, utilizada geralmente para operações numéricas, de modo prefixado (linha 2), em que o nome da função é seguido pelos seus argumentos, como também pode haver funções anônimas (linha 3) (HUDAK et al., 2007).

```
1. 4 `max` 7
2. max 4 7
3. (\x y -> max x y)
```

Como foi apresentado sobre linguagens funcionais, Haskell permite a aplicação de funções parcialmente, graças à técnica de *currying*.

```
sub :: (Num a) => a -> a -> a
sub a b = a - b
```

A função ***sub*** recebe um argumento, chamado de **a**, então retorna uma função que recebe outro argumento, chamado **b** que retorna o valor da subtração entre os dois argumentos.

Funções em Haskell tem uma peculiaridade especial, elas sempre devem retornar alguma coisa porque elas são consideradas expressões. Podem ser implementadas de diversas

formas, incluindo *pattern matching*, em que será executada a função de acordo com o padrão dos seus argumentos (LIPOVAČA, 2011).

```
fatorial 0 = 1
fatorial n = n * fatorial (n - 1)
```

Outra estrutura bastante utilizada são os guardas, muito semelhante a um if de linguagens imperativas, seu retorno depende de uma avaliação dos parâmetros (LIPOVAČA, 2011).

```
imcCla imc
    | imc <= 18.5 = "Abaixo"
    | imc <= 25.0 = "Normal"
    | otherwise   = "Acima"
```

Algo que é muito similar também à utilização de expressões *case*, se avalia e executa determinada ação dependendo desse valor (LIPOVAČA, 2011).

```
describeList xs = "A lista é " ++ case xs of [] -> "vazia."
                                             [x] -> "unitária."
                                             xs -> "longa."
```

Programas em Haskell podem ser divididos em módulos, deixando o projeto mais organizado e facilitando a reutilização de código. Eles encapsulam tipos de dados, funções, etc., exportando o que se faz necessário. Então os módulos podem ser reaproveitados em novos projetos. Haskell conta com alguns módulos disponíveis, como o Data.List, Data.Char, Data.Map e Data.Set, para trabalhar com listas, strings, mapas chave-valor e conjuntos, respectivamente, que podem ser utilizados realizando apenas a sua importação (LIPOVAČA, 2011).

Como citado anteriormente sobre linguagens funcionais, elas podem ser interpretadas, como também compiladas, com o geração de um código binário pelo compilador. Para Haskell, o compilador mais conhecido é o GHC, ele vem acompanhado de um interpretador interativo, o *ghci*.

Como qualquer outra linguagem de programação, oferece a possibilidade de comentários ao longo do código com o objetivo de documentar. Haskell ainda conta com uma vasta documentação no próprio site da linguagem, além de ferramentas para auxiliar o programador na documentação de seus projetos, como o Haddock[9], por exemplo. Está

[9] www.haskell.org/haddock/

disponível também para o programador o Hackage[10], uma central de pacotes, em que programadores podem utilizar pacotes desenvolvidos por outros programadores, assim como também publicar suas próprias bibliotecas.

Observando superficialmente os pilares e demais características apresentadas aqui, pode não se imaginar o poder que essa linguagem pode oferecer ao programador, mas com um estudo mais a fundo esse poderio se torna surpreendente, o que foi comprovado durante a pesquisa para desenvolvimento deste trabalho. Por isso, a escolha da linguagem Haskell no desenvolvimento do projeto detalhado mais à frente, demonstrando que é possível a implementação de softwares robustos utilizando interface gráfica e tudo mais, a partir de uma linguagem de programação funcional, que inicialmente pode aparentar fragilidades e deficiências em alguns aspectos, pela ideia inicial de uma linguagem de terminal de comandos.

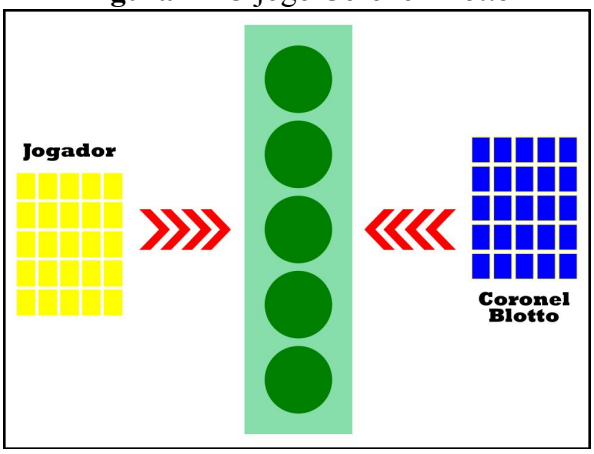
2.4 Coronel Blotto

Nesta Seção são debatidos o jogo que será estudado e implementado, suas características e definições, bem como a sua ligação com a Teoria dos Jogos, como ele se comporta dentro dessa área e alguns pontos pertinentes para a implementação do mesmo.

O jogo Coronel Blotto, também conhecido como jogo *divide a dollar* ou *Blotto game*, foi sugerido inicialmente por Borel (1921) e resolvido na sua forma clássica. Uma partida é formada por dois jogadores, o jogador e o seu opositor, o coronel Blotto, cada qual com um número de tropas igual ou diferente; em alguns casos o coronel Blotto tem um número maior. A fim de aumentar a dificuldade para o jogador é disposto também um mapa, ou não, dependendo da implementação do jogo, com os territórios que estão disponíveis para os jogadores disputarem. A versão inicial do jogo, proposta por Borel continha três territórios. O jogo vai consistir na distribuição, pelas duas partes, de suas tropas de soldados para os campos de batalha (MAIOLI, 2015). A Figura 4 demonstra uma visão geral do jogo.

[10] hackage.haskell.org/

Figura 4 - O jogo Coronel Blotto



Fonte: Elaborada pelo autor.

Ao final da distribuição é realizada a contagem do número de soldados de cada coronel nos campos de batalha, em que cada campo pertencerá ao coronel que alocar mais soldados, contabilizando por campo 01 ponto (dependendo da versão implementada do jogo, cada campo poderá oferecer um valor variável, levando em consideração características daquele território). Vale lembrar que essa alocação de tropas de cada jogador é feita sem o conhecimento por parte do outro, e como seria no mundo real, o maior número de tropas derrota o oponente e fica com o território.

No fim é contabilizada a quantidade de vitórias, derrotas e empates de cada um dos jogadores; o jogador que ficar com mais territórios sob seu comando é o vencedor do jogo. O *payoff* pode ser considerado de várias formas, a depender da escolha do autor.

Na forma clássica, citada por Maioli (2015), o *payoff* pode ser calculado da seguinte forma para o jogador e também para o coronel Blotto: sendo a soma dos resultados de cada campo de batalha, em que o resultado de cada campo é calculado dessa maneira: (+1) em caso de vitória, (0) em caso de empate e (-1) em caso de derrota. Para mais de uma rodada basta repetir o processo.

É um jogo que tem utilidade em diversas áreas em que são empregados conceitos da Teoria dos Jogos, como na Economia e na Política. Na Economia um exemplo interessante seria um estudo de Szentes e Rosenthal (2003) citados por Maioli (2015), em que de forma resumida, eles idealizam uma situação de leilão na qual existem três objetos ofertados a dois participantes, eles registram suas ofertas simultaneamente sem que o oponente saiba, quem oferecer o maior valor para cada objeto fica com o mesmo. O vencedor do leilão seria o que adquire dois ou mais objetos.

Myerson (1993) citado por Maioli (2015) e também por Kovenock e Roberson (2015), traz em seu artigo um estudo utilizando o jogo para simular a distribuição de recursos econômicos em campanhas eleitorais, provando com esse estudo que não existe um equilíbrio de Nash para as estratégias disponíveis puramente, comprovando também que não existe uma estratégia sempre vitoriosa nesse contexto, por outro lado, adicionando as estratégias mistas o equilíbrio de Nash se torna possível.

Após essa breve e simplificada descrição do jogo, temos uma ideia geral de como o mesmo é executado, com isso podemos abstrair algumas características e vertentes. A principal questão em que gira esse jogo clássico é em torno de como organizar uma melhor distribuição de tropas entre os campos para se conseguir um maior número de territórios, o que leva em consideração um grande número de variáveis, como número de territórios, quantidade de soldados, etc.

As regras que dizem respeito a pontuação ou questões relacionadas a especificações de valores são bem flexíveis, visto que a ideia principal do jogo não é apenas retratar uma situação específica em si, o jogo, mas sim debater uma questão análoga, abstraindo pormenores. O jogo é idealizado em sua forma clássica, não há nenhum empecilho impedindo que sejam feitas algumas adaptações para melhorar o entendimento de alguns conceitos.

Esse jogo foi escolhido para ser debatido aqui por ele ter a sua devida importância na área da Teoria dos Jogos, servindo como base para muitos autores realizarem uma discussão de casos específicos e apresentação de conceitos da área. Como já foi debatido anteriormente os tipos de classificação dos jogos e também uma rápida descrição do jogo Coronel Blotto, com suas regras, agora serão discutidas algumas classificações desse jogo dentro da área da Teoria dos Jogos.

Considerando a versão clássica do jogo, se trata de um jogo de **soma zero**, visto que o ganho do jogador é diretamente impactado pela perda do jogador adversário. A soma dos *payoffs* resultantes é igual a zero. É um jogo **não-cooperativo** já que a essência do jogo é escolher uma estratégia que minimize o ganho do jogador oponente e consequentemente maximize os próprios; não há nenhum tipo de colaboração entre os jogadores.

É um jogo **simultâneo**, pois os jogadores devem dividir suas tropas de soldados ao mesmo tempo, sem que um tenha conhecimento da estratégia adversária, assim como seria no mundo real; é de **informação imperfeita**, os jogadores não têm informações prévias do jogo, como também referentes à estratégia adversária.

Se classifica como um jogo **estático** pelo fato de haver apenas uma interação entre as estratégias por partida, por outro lado ele é **simétrico** levando em consideração que se as estratégias forem invertidas as suas recompensas também serão invertidas, haja vista que os dois jogadores têm o mesmo número de tropas disponíveis.

Ele se classifica também como um jogo de **informação completa**, já que os jogadores têm completo conhecimento das regras e possíveis resultados da utilização das estratégias disponíveis para eles. Uma última classificação dentre as elencadas ao longo deste texto é que o Coronel Blotto se encaixa como um jogo **indeterminado**, em que, já que o jogador não tem uma garantia de conseguir no mínimo um empate sempre, o adversário sempre vai agir de uma forma imprevisível.

É visível que realizando pequenas alterações na estrutura do jogo e utilizando versões com modificações específicas pode-se obter diferentes classificações. Isso se deve ao fato de que as classificações de jogos pela Teoria dos Jogos têm naturezas passíveis de mudanças; são empregadas correspondendo a questões que são de fácil identificação. Por exemplo, alterando a forma como o ganho é distribuído ele pode se tornar um jogo de soma não nula, uma versão específica.

Assim, as regras do jogo que serão utilizadas neste trabalho apresentam alguns pontos específicos, para que o jogo decorra da forma mais didática possível. O principal foco aqui é a demonstração das características definidas na Teoria dos Jogos. A seguir são elencadas as regras que serão adotadas na implementação e execução dessa versão do jogo:

1. Quantidade de soldados do jogador e do Coronel Blotto (serão consideradas iguais): será utilizada uma quantidade de tropas variável, com 100, 150 ou 200 soldados;
2. Quantidade de campos de batalhas: também será possível variar o número de campos de batalha entre 03, 04 e 05 campos;
3. Possibilidade de partidas: o jogador poderá jogar quantas partidas desejar;
4. Em relação ao *payoff* serão contabilizadas apenas as vitórias de cada jogador, empates e derrotas não vão interferir no placar.

Com essas definições, que de certa forma serviram de levantamento de dados para o processo de desenvolvimento, como requisitos funcionais e não-funcionais, pode-se iniciar o processo de desenvolvimento, descrito na Seção de metodologia.

2.5 Trabalhos correlatos

Foram encontrados alguns trabalhos correlatos com os assuntos apresentados de forma geral com este trabalho.

Caldeira (2010) apresenta de uma maneira geral uma ideia muito parecida com o trabalho atual, demonstrando uma ferramenta computacional construída na linguagem Java para web, utilizando como exemplos jogos que são conhecidos na Teoria dos Jogos para repassar alguns conceitos a respeito da teoria, como jogos cooperativos e jogos não-cooperativos.

O trabalho de Furtado e Santos (2002) se relaciona de certa forma com a implementação da interface gráfica, onde eles relatam uso de ferramentas para desenvolvimento de interfaces gráficas e a criação de uma ferramenta com esse objetivo, criada utilizando Haskell e para ser utilizada nela.

3 METODOLOGIA

Nesta Seção será abordado o processo de implementação do jogo Coronel Blotto, utilizando a linguagem de programação Haskell. Discutindo desde a parte de análise até o produto concluído.

3.1 Processo de desenvolvimento

Para implementação do jogo Coronel Blotto foram estudados e analisados o funcionamento de diversos recursos, que pudessem contribuir e entregar um desenvolvimento satisfatório do jogo, agilizando o processo de desenvolvimento já que o tempo para implementação do mesmo era de certa forma limitada. Nesta Seção é relatado o processo de desenvolvimento, partindo da ideia até o projeto concluído.

Para o desenvolvimento do jogo foram utilizados alguns conceitos da modelagem de desenvolvimento ágil, visto que os detalhes do jogo final não estavam bem definidos no início do projeto e foram sendo clarificados no decorrer das reuniões com o orientador.

De acordo com Sommerville (2011), o desenvolvimento ágil possui algumas características específicas, juntamente com seus objetivos. Entre os pontos na parte de desenvolvimento baseados no manifesto ágil se destacam a forte participação do orientador nesse processo, o projeto tem uma entrega incremental ao longo do desenvolvimento, preferindo a interatividade entre as pessoas, podendo assim corresponder melhor às mudanças, enfatizando mais o desenvolvimento de funcionalidades.

Esse método de desenvolvimento oferece uma maior aceitação e adaptação às mudanças, a implementação se dá de maneira incremental, aos poucos funcionalidades são integradas, facilitando as mudanças, caso sejam necessárias. É uma metodologia que busca a simplicidade na implementação e que valoriza as habilidades da equipe de desenvolvimento com o intuito de melhorar a qualidade do desenvolvimento, sem ficar preso a processos.

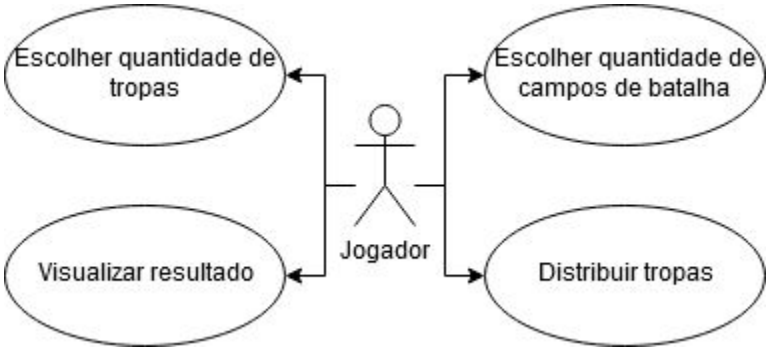
Foi, portanto, um método identificado como ideal para a atual abordagem de desenvolvimento, bem como seus objetivos, entre outras variáveis envolvidas. O projeto foi desenvolvido utilizando um cronograma curto para entrega, então algumas modificações se fizeram necessárias para conclusão de uma versão Beta.

Após a definição do jogo que seria implementado, foram realizadas algumas modelagens com o intuito de conduzir o processo de desenvolvimento da maneira mais

simples possível, visto que seria necessário obter conhecimentos para suprir a demanda que o projeto necessitava.

Como parte do planejamento inicial foram definidas questões específicas do jogo, foi possível definir um diagrama de caso de usos (Figura 5) identificando requisitos que a implementação do jogo deveria oferecer ao usuário para melhorar a sua experiência.

Figura 5 - Diagrama de casos de uso do jogo

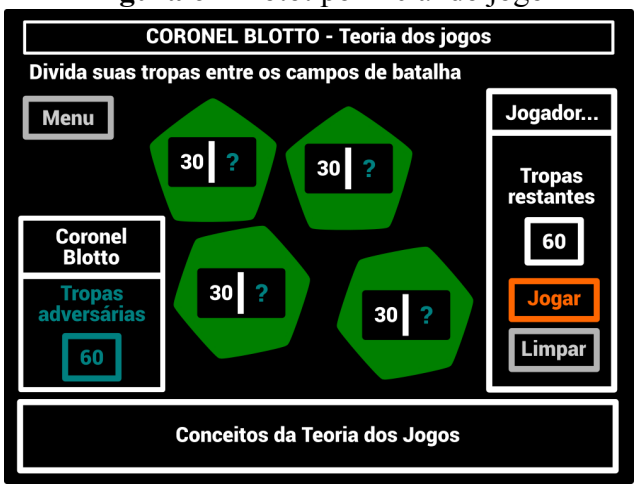


Fonte: Elaborada pelo autor.

Seguindo o processo de desenvolvimento, com reuniões semanais, o orientador participando efetivamente, com sugestões de melhorias, funcionalidades, etc., foi implementada a lógica geral do jogo, chegando a uma versão funcional em modo de linha de comando para melhorias e correções de erros provenientes do desenvolvimento, relatados pelo orientador durante testes.

Seguindo com o planejamento para implementação de novas funcionalidades iniciou-se a modelagem para desenvolvimento de uma interface gráfica para o jogo, um dos requisitos que foram propostos inicialmente. Chegamos a um protótipo básico (Figura 6), mas que agrupava todas as funcionalidades requeridas e uma sugestão da disposição dos elementos na tela, o que é muito importante para receber a opinião do orientador, melhorando os pontos em que ele achar necessário.

Figura 6 - Protótipo inicial do jogo



Fonte: Elaborada pelo autor.

A partir da criação do protótipo para a interface gráfica (apresentado na Seção 3.2), pôde ser iniciado o processo de pesquisa para suprir essa demanda, visto que o conhecimento nessa área não era suficiente.

O desenvolvimento da interface gráfica para o jogo prezou pela organização do código fonte, como uma boa prática da programação, separando em módulos Haskell, sempre que necessário, buscando assim utilizar os recursos fornecidos pela linguagem para melhorar a qualidade do desenvolvimento.

Após concluir a implementação da interface, foi realizada a integração com a parte lógica do jogo, que já havia sido desenvolvida, ajustando alguns detalhes na interface de comunicação entre as duas partes, lançando assim uma versão Beta da aplicação.

O código fonte desenvolvido está disponível no repositório[11] do autor, juntamente com as *releases* que foram lançadas e a sua documentação[12].

3.2 Interface gráfica

A utilização de Haskell para desenvolvimento de um projeto desse porte à primeira vista leva a se pensar apenas em uma tela preta com entrada de comandos pelo teclado, como foi implementada a versão inicial e também como se dava a interação com programas de computador algumas décadas atrás. Um requisito inicial proposto pelo orientador foi o de

[11] github.com/juakacc/colonel-blotto
[12] github.com/juakacc/colonel-blotto/blob/master/docs/guide.rst

implementar o jogo com uma interface gráfica, simples, mas que facilitasse a interação do jogador com o jogo, melhorando a sua usabilidade.

Foi um desafio para a equipe de desenvolvimento, pois as ferramentas disponíveis com esse intuito são um tanto quanto limitadas. Foi então levantada uma série de pacotes/bibliotecas que poderiam contribuir nessa parte do desenvolvimento, entre as principais estão: Gloss[13], Gtk2Hs[14], VTY[15], BRICK[16]. Após um estudo mais aprofundado de cada uma delas foram levantados seus pontos positivos e negativos a fim de decidir a que melhor se encaixaria no projeto, com a maior simplicidade possível e dentro do tempo, mas que suprisse as necessidades propostas inicialmente.

Foi escolhida a BRICK para desenvolvimento da GUI (*Graphical User Interface*), pela sua simples forma de implantação, mantendo, mesmo assim, um poderosíssimo potencial para criação de interfaces gráficas, utilizando a própria janela do terminal, o que remete à simplicidade da mesma, mas proporcionando uma interação avançada com o usuário, como interação com o mouse, formulários, elementos de *feedback*, elementos para design, entre outros. Ela tem uma curva de aprendizagem consideravelmente baixa, o que teve peso em sua escolha para implantação no projeto.

Além da sua simplicidade e o poder que oferece ao programador, é um pacote que apresenta uma documentação bastante abrangente, juntamente com um fórum de discussão ativo na comunidade, com os membros que estão sempre dispostos a ajudar quem está com dificuldade sobre a biblioteca, o que ajuda muito novos programadores com dúvidas que surgem ao longo do processo de aprendizagem. Conta também com uma série de programas de exemplo contemplando diversos recursos que servem de base para implementação de ações específicas.

Ele foi desenvolvido baseado em outro pacote, o Vty-ui, que acabou depreciado em favor do BRICK, pela sua grande aceitação pela comunidade. O VTY fica responsável por prover e tratar eventos de entrada e saída do terminal, enquanto que o BRICK se encarrega de gerenciar os requisitos restantes, fornecendo para o programador um alto nível de abstração para a programação. Nesse contexto, ela requer apenas que o programador implemente funções para tratar os eventos de I/O (eventos de entrada e saída) que serão lançados e uma

[13] http://hackage.haskell.org/package/gloss
[14] https://wiki.haskell.org/Gtk2Hs
[15] http://hackage.haskell.org/package/vty
[16] http://hackage.haskell.org/package/brick

função para “desenhar” a tela da aplicação. Esses são os requisitos mínimos para o funcionamento do BRICK no projeto, outras questões são de implementação facultativa, como a estilização da interface, cores, alinhamento, preenchimento, etc. Ele fornece um extenso conjunto de primitivas declarativas, que simplificam muito o processo de implementação, ao contrário do que é exigido por outras bibliotecas, que aumentam o conhecimento necessário e tornam mais complexa a legibilidade do código.

Toda a aplicação trabalha mantendo um estado, diferente daquele debatido na Seção 2.2, sobre programação funcional, que em Haskell pode ser representado como um *type,* armazenando informações do estado atual no decorrer da execução quando os eventos vão sendo gerados e a função responsável por esse tratamento é invocada. Com isso ela deve realizar as alterações necessárias no estado atual, dependendo do evento e da forma como foi gerado. Toda vez que é realizada alguma alteração no estado a função responsável pela renderização da interface é chamada, desenhando uma nova interface para o novo estado da aplicação. Esse é o *loop* em torno do qual gira uma aplicação que utiliza BRICK para implementação da interface gráfica. Todas essas funções envolvidas são definidas em outra função, que deve ser chamada a partir da função *main.*

A BRICK requer que sejam determinados quais tipos representarão o estado, os eventos e o nome de recursos (que seriam basicamente elementos da interface que têm interação, que podem disparar eventos, criados por meio de um type), na função chamada na função principal, no caso do projeto a função *app*. Os tipos definidos foram: *AppState*, *AppEvent* e *Name*, respectivamente.

Como a aplicação demandou basicamente quatro telas, foram definidas quatro funções de desenho. A função que será executada é escolhida pela função *drawUI* que decide baseada em uma *flag* que sinaliza em qual tela o estado se encontra. Através do type *App* que é oferecido pela biblioteca foi definida também a função para manipular os eventos, *handleEvent*. Ela trabalha da mesma forma que a função de renderização, dependendo de qual tela esteja selecionada, invoca a função correspondente. Como adicional pode ser fornecido um *appChooseCursor* que tem como objetivo definir onde será localizado o cursor. Existe também o *appAttrMap,* que permite ao programador mapear atributos de elementos da interface, tornando esse trabalho bem mais legível.

O loop segue executando indefinidamente até que algum evento de saída do programa seja executado pelo usuário. Esses eventos também devem ser implementados pelo

programador, decidindo qual a melhor maneira para fazer isso, dependendo da aplicação, no projeto foi adotado o clique no botão de sair ou utilizando as teclas de atalhos, disponíveis no lado esquerdo da aplicação.

3.3 Dificuldades enfrentadas

Ao longo do processo de desenvolvimento foram encontrados alguns problemas com relação às questões de codificação, o que é normal em qualquer projeto de software, levando em consideração a utilização de novas tecnologias, adaptação de paradigmas diferentes, como o que foi proposto, o processo de aprendizagem de uma nova metodologia, todos esses fatores, entre outros, têm um peso sobre o processo de implementação, mas que trabalhados da maneira correta puderam ser contornados com sucesso.

Como Haskell é uma linguagem puramente funcional, ela trabalha com uma ideia de *Monad*, algo que pode ser explicado de maneira geral como um valor envolvido por um contexto. Foi a principal dificuldade encontrada ao longo do desenvolvimento. Haskell trabalha com funções puras, ou seja, é possível afirmar que para uma determinada função tem-se o retorno esperado, a definir pela entrada; já funções impuras são aquelas que não se pode usar essa afirmativa, mesmo conhecendo a entrada. Todas as funções de I/O em Haskell trabalham de forma impura, entrada de dados pelo teclado, como também a geração de números randômicos, que foi necessária para determinação da estratégia utilizada pelo Coronel Blotto no jogo. A própria documentação elege uma forma de amenizar esses efeitos e como trabalhar de forma pura, levando em conta que o programador tenha conhecimento dos dados que estão sendo trabalhados.

Outra dificuldade enfrentada foi a possibilidade, por meio da biblioteca BRICK, para desenho de algumas formas geométricas, como também para preenchimento. Não é fornecido pela biblioteca, mas pode ser resolvido utilizando outras formas.

Até o momento, a BRICK não oferece suporte a sistemas Windows, pelo motivo que é altamente dependente do VTY, que por sua vez depende do Unix e não oferece uma compatibilidade com o sistema operacional citado. Esta é uma limitação determinada pela biblioteca BRICK.

4 CONCLUSÃO

Foram apresentados os principais conceitos da Teoria dos Jogos, conceitos da programação funcional demonstrando como são implementados em uma linguagem de programação real, como a linguagem Haskell, por fim a explanação de um exemplo de jogo que utilizou a Teoria dos Jogos com a principal finalidade de demonstrar o poder da programação funcional por meio da implementação do jogo Coronel Blotto, juntamente com o desenvolvimento da GUI para melhorar a experiência do usuário e assim tentar repassar conceitos da Teoria dos Jogos.

Com a apresentação dos assuntos abordados, foram notáveis os proveitos que a Teoria dos Jogos pode oferecer nas várias áreas de sua aplicação (Economia, Política, etc), como em questões que envolvam decisões de interesses entre pessoas.

É considerável a produtividade que linguagens funcionais podem trazer para o dia a dia do programador, com os seus diversos pontos positivos, que muitas vezes são julgadas antecipadamente sem qualquer embasamento.

4.1 Futuras versões

Visto que os requisitos inicialmente propostos foram atendidos satisfatoriamente, em futuras versões do jogo serão realizadas melhorias, em relação à usabilidade e à persistência de dados armazenados ao longo das jogadas, criação de um *ranking* ou algo do tipo, com uma melhor definição da pontuação, como a verificação da possibilidade de torná-lo um jogo *multiplayer*. Também verificar a possibilidade de alguma implementação ou versão para o sistema operacional Windows, disponibilizando o jogo para mais usuários, aceitando colaborações no repositório oficial ou por *e-mail*[17].

[17] juakacc@gmail.com.

REFERÊNCIAS

ALLEN, Christopher; MORONUKI, Julie. **Haskell programming from first principles**. New York: [s. n.], 2016.

ALMEIDA, Alecsandra Neri de. **Teoria dos Jogos: As origens e os fundamentos da Teoria dos Jogos**. UNIMESP - Centro Universitário Metropolitano de São Paulo, 2006.

ARVELOS, Geraldo de. **A TEORIA DOS JOGOS E O ENSINO DE OTIMIZAÇÃO**. 2009. Dissertação de Mestrado - Pontifícia Universidade Católica de Minas Gerais, Belo Horizonte/MG, 2009.

AZEVEDO, André Gomma de. **Estudos em Arbitragem, Mediação e Negociação**. Vol 2. Brasília: Grupos de Pesquisa, 2003.

BARENDREGT, H. P.; BARENDSEN, E. **Introduction to lambda calculus** in Aspenæs Workshop on Implementation of Functional Languages, Göteborg, 1988.

BÊRNI, Duilio de Avila. **Teoria dos Jogos.** São Paulo: Reichmann e Affonso, 2004.

BOREL, Émile. *The Theory of Play and Integral Equations with Skew Symmetric Kernels*. 1921.

BORTOLOSSI, Humberto José; GARBUGIO, Gilmar; SARTINI, Brígida. **UMA INTRODUÇÃO À TEORIA ECONÔMICA DOS JOGOS**. 2017.

CALDEIRA, Rosane. **Interface Gráfica no Contexto de Teoria dos Jogos sob a forma de Java Applets**. 2010. Dissertação de Mestrado - UNESP, Ilha Solteira/SP, 2010.

CALÔNEGO JUNIOR, Nivaldi. **Uma Abordagem Orientada a Objetos de uma Ferramenta de Auxílio à Programação Paralela**. 1997. Tese de Doutorado - USP, São Carlos/SP, 1997.

CÂMARA, Samuel Façanha. **Teoria dos jogos** - Florianópolis: Departamento de Ciências da Administração/UFSC, 2011.

CARVALHO, Lucas Augusto Montalvão Costa. **Abordagens de Teoria dos Jogos para modelagem de Sistemas de Recomendação para Grupos**. 2013. Dissertação de Mestrado - UFS, São Cristovão/SE, 2013.

FELICIANO, Léa Paz da Silva. **Teoria dos Jogos: Uma nova proposta para o ensino médio**. 2007. Dissertação de mestrado - PUC/SP, São Paulo/SP, 2007.

FIANI, Ronaldo. **Teoria dos Jogos:** com aplicação em Administração, Ciências Sociais e Economia. 2. ed. Rio de Janeiro: Elsevier, 2006.

FIANI, Ronaldo. **Teoria dos Jogos:** Com aplicações em Economia, Administração e Ciências Contábeis. 3. ed. Brasil: Elsevier, 2009.

FURTADO, André Wilson Brotto; SANTOS, André Luís de Medeiros. **FunGEn: A Game Engine for Haskell**, 1st Brazilian Workshop in Games and Digital Entertainment (Wjogos2002), 2002.

FRANCEZ, David Jonnes. **Uma introdução à Teoria dos Jogos**. 2017. Dissertação de Mestrado - UFSC, Florianópolis/UFSC, 2017.

GUTH, Werner; SCHMITTBERGER, Rolf; SCHWARZE, Bernd. An experimental analysis of ultimatum bargaining. **Journal Of Economic Behavior & Organization.** Germany, p. 367-388. dez. 1982.

HASKELLWIKI. **The Haskell Programming Language**. Disponível em: https://wiki.haskell.org/Haskell. Acesso em: 30 out. 2019.

HUDAK, Paul. et al. **A History of Haskell:** Being Lazy With Class. Proceedings of the Third ACM SIGPLAN Conference on History of Programming Languages. San Diego, California, 2007.

JIANYA, Zheng. **Uma Investigação de Relacionamentos Baseados na Competição entre Stakeholders no Comércio Eletrônico Utilizando Teoria dos Jogos**. 2016. Tese de Doutorado - UnB, Brasília/DF, 2016.

JUNKER, Joseph. **Currying and Uncurrying in JavaScript and Flow**. Disponível em: https://link.medium.com/hLft1PNuj1. Acesso em: 02 nov. 2019.

KELLY, Anthony. **Decision Making using Game Theory:** An introduction for managers. Cambridge: Cambridge University Press, 2003.

KOVENOCK, Dan J.; ROBERSON, Brian. **Generalizations of the General Lotto and Colonel Blotto Games**. 2015. ESI Working Paper 15-07. Disponível em <http://digitalcommons.chapman.edu/esi_working_papers/156>. Acesso em 25 out. 2019.

LANZOTTI, Carla Regina. **Plataforma computacional de auxílio à comercialização de energia elétrica**. 2006. Tese de Doutorado - Universidade Estadual de Campinas, Faculdade de Engenharia Mecânica, Campinas/SP, 2006.

LIPOVAČA, Miran. **Learn You a Haskell for Great Good!:** A Beginner's Guide. Slovenia: no Starch Press, 2011.

MAIOLI, Alan Carlos. **Quantização do Jogo Coronel Blotto**. 2015. Dissertação de Mestrado - UFF, Volta Redonda/RJ, 2015.

MCCARTHY, John. **Recursive functions of symbolic expressions and their computation by machine**. Massachusetts Institute of Technology, Cambridge. Communications of the ACM. 1960.

MYERSON, Roger B. **Analysis of Conflict.** United States Of America: Harvard University Press, 1991.

O'SULLIVAN, Bryan; GOERZEN, John; STEWART, Don. **Real World Haskell.** United States Of America: O'reilly, 2008.

PAIVA, Jonas Alves de. **APPLICATION OF DECISION THEORY TO SOLVE GAME THEORY PROBLEMS**. 2012. Tese de Doutorado - UFPE, Recife/PE, 2012.

PEREIRA, Meyrele Alves. **Teoria dos Jogos: uma ferramenta para provas no cálculo proposicional**. 2006. Trabalho de Conclusão de Curso - Centro Universitário de João Pessoa, João Pessoa/PB, 2006.

PINDYCK, Robert S.; RUBINFELD, Daniel L. **Microeconomia.** 6. ed. São Paulo: Pearson Prentice Hall, 2006. Tradução de Eleutério Prado e Thelma Guimarães.

REDAÇÃO NUBANK (Brasil). **O que é programação funcional e o que isso tem a ver com o Nubank?** 2019. Disponível em: https://blog.nubank.com.br/programacao-funcional-o-que-e-relacao-nubank/. Acesso em: 15 nov. 2019.

RIBEIRO, Abner. et al. **Dilema dos Prisioneiros e Equilíbrio de Nash**. FATEC Sorocaba, 2013. Disponível em <http://efinfatec.blogspot.com/2013/12/artigo-dilema-dos-prisioneiros-e.html>. Acesso em 20 jun. 2019.

RIBEIRO, Vitor Filincowsky. **Decisão colaborativa com utilização de Teoria dos Jogos para o sequenciamento de partidas em aeroportos**. 2013. Dissertação de Mestrado - UnB, Brasília/DF, 2013.

ROY, Sankardas; ELLIS, Charles; SHIVA, Sajjan; DASGUPTA, Dipankar; SHANDILYA, Vivek; WU, Qishi. **A Survey of Game Theory as Applied to Network Security**. 43rd Hawaii International Conference on System Sciences (HICSS). Hawaii, USA, January, 2010.

SARTINI, Alexandre Sartini; GARBUGIO, Gilmar; BORTOLOSSI, Humberto José; SANTOS, Polyane Alves; BARRETO, Larissa Santana. **Uma introdução a Teoria dos Jogos**. II Bienal da Sociedade Brasileira de Matemática, UFBA, 2004.

SEBESTA, Robert W. **Conceitos de Linguagens de Programação.** 9. ed. Porto Alegre: Bookman, 2011. Tradução de Eduardo Kessler Piveta.

SILVEIRA, Luciana Torrezan. **Procedimento para análise de decisão quanto à prevenção de doenças em animais: uma aplicação da Teoria dos Jogos**. 2008. Tese de Doutorado - USP, Piracicaba/SP, 2008.

SCHMITT, Fátima Ap. Benthien da Silva. **Utilização da Programação Funcional no Mundo dos Blocos Geométricos**. 2003. Dissertação de Mestrado - UFSC, Florianópolis/SC, 2003.

SOMMERVILLE, Ian. **Engenharia de Software.** 9. ed. São Paulo: Pearson Prentice Hall, 2011. Tradução de Kalinka Oliveira e Ivan Bosnic. Revisão Técnica do Prof. Dr. Kechi Hirama.

SOUSA, Bruno Jefferson de; DIAS JÚNIOR, José Jorge Lima; FORMIGA, Andrei de Araújo. **Introdução a Programação**. João Pessoa: Editora da UFPB, 2014.

TADELIS, Steve. **Game Theory:** An Introduction. United States Of America: Princeton University Press, 2013.

TUCKER, Allen B.; NOONAN, Robert E. **Linguagens de Programação:** Princípios e Paradigmas. 2. ed. Porto Alegre: Amgh, 2010. Tradução de Mario Moro Fecchio e Acauan Fernandes. Revisão Técnica de Eduardo Marques e Márcio Merino Fernandes.

UCHÔA, Thiago Montenegro. **Controle de Potência de Transmissão em VANETs: Uma abordagem utilizando Teoria dos Jogos**. 2015. Dissertação de Mestrado - UFPE, Recife/PE, 2015.

WARBURTON, Richard. **Object-Oriented vs. Functional Programming.** United States Of America: O'reilly, 2016.

XAVIER, Robson Duarte. **Paradigmas de desenvolvimento de software: comparação entre abordagens orientada a eventos e orientada a notificações**. 2014. Dissertação de Mestrado - UTFPR, Curitiba/PR, 2014.

APÊNDICE A – JOGOS EXEMPLOS

Aqui serão debatidos alguns jogos que são trabalhados na Teoria dos Jogos e que foram utilizados como exemplo neste trabalho.

- Batalha dos Sexos

A Batalha dos Sexos pode ser descrita como um evento em que um casal deseja sair para descontrair e os dois possuem diferentes preferências em relação ao lugar aonde querem ir, revelando um pouco de incompatibilidade e gostos pessoais comuns. Um dos exemplos mais utilizados é o do cinema e do futebol, em que a mulher prefere ir ao cinema, por outro lado o homem deseja ir assistir ao jogo de futebol. É sabido que cada um tem uma maior satisfação quando está frequentando o evento de sua preferência e que os dois desejam permanecer juntos no evento que de alguma forma será escolhido. Uma possível modelagem para o jogo seria a seguinte:

J = {mulher, homem}, E-mulher = {cinema, futebol}, E-homem = {cinema, futebol}, E = {(cinema, cinema), (cinema, futebol), (futebol, cinema), (futebol, futebol)}

Sendo o "**J**" o símbolo referente ao conjunto de jogadores, "**E-mulher**" a representação das estratégias puras da mulher, "**E-homem**" a representação das estratégias puras do homem e "**E**" simbolizando o conjunto dos pares de todas as estratégias disponíveis para os jogadores.

Atribuindo a essas escolhas (pares ordenados) determinados valores, que são chamados de *payoffs*, temos a representação desse jogo na forma normal, como pode ser visto no Quadro 3.

Quadro 3 – A Batalha dos Sexos na forma normal

	Mulher	
Homem	Cinema	Futebol
Cinema	(1, 2)	(0, 0)
Futebol	(0, 0)	(2, 1)

Fonte: Pindyck e Rubinfeld (2006, com adaptações).

Na matriz de *payoff* sabe-se que as linhas representam as opções de escolha do homem e as colunas as da mulher. O par ordenado (1,2) representa o *payoff* da escolha (Cinema, Cinema), em que o primeiro indica a recompensa do homem e o segundo a recompensa da mulher.

A partir das informações disponíveis acima, sabe-se que A Batalha dos Sexos é um jogo que possui dois jogadores, duas estratégias por jogador e apresenta dois equilíbrios de Nash com estratégias puras, que são as combinações de estratégia com maior ganho para os dois.

- Cara ou Coroa

Cara ou coroa, também conhecido como *matching pennies*, é uma versão do jogo entre dois jogadores e que é muito utilizado na Teoria dos Jogos. Nele cada um dos jogadores possui uma moeda. Sendo assim, as estratégias disponíveis para eles são: cara (C) ou coroa (K). Um dos jogadores ganha quando as faces das moedas coincidirem, ou seja, se forem (C, C) ou (K, K). Por outro lado, o adversário ganha se as faces das moedas forem diferentes, como é mostrado no Quadro 4. A escolha da opção é realizada antes do início da partida. Prosseguindo, as moedas são arremessadas para cima; ao aterrissarem são verificadas quais são as faces que estão viradas para cima, apontando o vencedor da rodada. Nas próximas rodadas as opções podem permanecer ou ser alteradas, isso fica a cargo dos jogadores. Uma representação desse jogo na forma normal pode ser considerada da seguinte maneira:

Quadro 4 - Cara ou Coroa na forma normal

	Jogador 2 (diferentes)	
Jogador 1 (iguais)	C	K
C	(+1, -1)	(-1, +1)
K	(-1, +1)	(+1, -1)

Fonte: Pindyck e Rubinfeld (2006, com adaptações).

Esse jogo é muito semelhante ao par ou ímpar, em que quando um jogador ganha, o outro perde exatamente aquilo que o outro jogador ganhou, assim como é mostrado acima, na matriz de recompensa.

- Pedra-papel-tesoura

Pedra-papel-tesoura é um jogo muito conhecido no mundo todo, onde possui diferentes variações e nomes, como por exemplo *jokenpô* ou *Rock-paper-scissors*. Se tratando do jogo em sua forma simples e original, geralmente é jogado com duas pessoas. Apresenta três estratégias para cada jogador, sendo elas: pedra, papel ou tesoura. No jogo, os jogadores **A** e **B** se posicionam um de frente para o outro com as mãos posicionadas para trás ou com a mão em forma de punho fechado. Após repetirem o nome do jogo simultaneamente exteriorizam suas escolhas, com um símbolo determinado para cada escolha, pedra (punho fechado), papel (mão aberta) e tesoura (dedos formando um "V" na horizontal).

Para se obter o resultado e definir o vencedor é usada a lógica natural, simulando o mundo real. É sabido que a pedra ganha de tesoura (a quebrando), o papel derrota a pedra (a cobrindo) e a tesoura consegue ganhar do papel (o cortando). Portanto, temos no Quadro 5 sua representação normal pela matriz de *payoffs*.

Quadro 5 - Pedra-Papel-Tesoura na forma normal

	Jogador B		
Jogador A	Pedra	Papel	Tesoura
Pedra	(0, 0)	(-1, +1)	(+1, -1)
Papel	(+1, -1)	(0, 0)	(-1, +1)
Tesoura	(-1, +1)	(+1, -1)	(0, 0)

Fonte: Feliciano (2007, com adaptações).

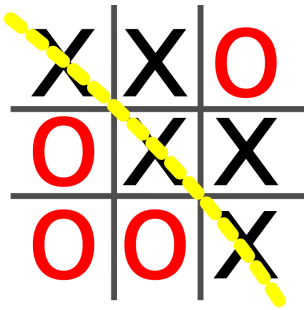
Podemos verificar que quando os dois jogadores optam pelas mesmas opções o jogo resulta em um empate, com a matriz de *payoffs* indicando um *payoff* de ganho zero para cada.

Assim como o jogo anterior, ele lembra um pouco o jogo do par ou ímpar, pois dadas as suas opções de escolha, cada jogador deve optar por uma de suas estratégias e apresentá-las ao mesmo tempo, dado que é um jogo de movimentação simultânea.

- Jogo da Velha

O tão famoso e célebre jogo da velha não poderia deixar de ser falado nessa lista de jogos debatidos pela Teoria dos Jogos, ele também é conhecido como *tic-tac-toe*, X's e O's, sendo um jogo bastante conhecido pelo público em geral, desde crianças até os mais velhos. É jogado entre dois jogadores, em que basicamente um deles é representado pelo "**O**" e o outro pelo "**X**". O tabuleiro, podemos assim dizer, é formado por 9 posições (uma grade de 3x3 campos), que serão ocupados cada um por um único jogador, como podemos ver na Figura 7.

Figura 7 - Representação do Jogo da Velha



Fonte: Elaborada pelo autor.

O jogo é iniciado por um dos jogadores; decidido entre eles, é feita a alternância entre as jogadas. De forma sequencial, o objetivo de cada jogador é preencher uma das linhas com a forma que o representa, seja ela uma linha vertical, horizontal ou diagonal, ganhando desse modo o jogo. Quando nenhum dos jogadores consegue preencher a linha antes que o tabuleiro fique completamente preenchido, se considera que o jogo deu empate, ele é geralmente chamado de "velha" aqui no Brasil, que pode ser consequência direta da escolha da melhor estratégia por ambos os jogadores.

- Jogo do Ultimato

Jogo do Ultimato é um jogo muito utilizado para estudos econômicos, que também envolvem a mente humana, muito utilizado para aplicação da Teoria dos Jogos.

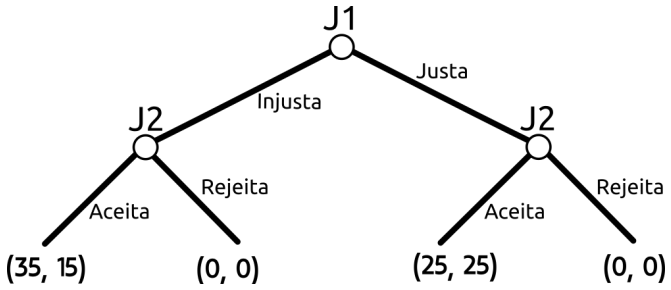
Consiste em uma relação com dois papéis principais, o proponente e o receptor. Previamente o proponente é avisado que pode fazer a divisão de uma oferta, já para o receptor é avisado que pode aceitar ou rejeitar a oferta proposta. O proponente então recebe uma determinada quantia para que possa dividir com um receptor; se o receptor aceitar o recebimento ambos os jogadores recebem a divisão realizada da quantia; se o receptor rejeitar esse recebimento, tanto o receptor quanto o proponente não recebem quantia alguma.

Dado que os jogadores conhecem perfeitamente as informações que são utilizadas no jogo, a questão principal gira em torno da decisão do receptor, que pode ou não aceitar o recebimento, independente da quantia, tornando o jogo imprevisível.

Sendo que a divisão proposta pelo primeiro jogador pode ser totalmente injusta, como por exemplo sugerir apenas 10% do total para o receptor, por outro lado, 10% de uma quantia pode ser melhor do que uma quantia zero. Também é sabido que a melhor divisão seria a igualitária (GUTH; SCHMITTBERGER; SCHWARZE*,* 1982).

Existem algumas variações, como o jogo do ditador, em que resumidamente o proponente permanece com a sua divisão, mesmo que o receptor rejeite a divisão, reduzindo a influência do receptor no resultado final do jogo. A Figura 8 mostra como fica o Jogo do Ultimato representado na forma extensa.

Figura 8 - Jogo do Ultimato na forma extensiva



Fonte: Elaborada pelo autor.

Na representação da forma extensa fica claro, como o Jogador 1 (J1) pode ofertar uma proposta justa ou não, considerando 50 como valor total, a oferta justa seria 25 para cada. o Jogador 2 (J2) sempre terá a opção de aceitar ou rejeitar a proposta, indicando os *payoffs* para cada jogador, localizado nas folhas da árvore. O primeiro elemento do par corresponde ao *payoff* do jogador 1 e o segundo ao jogador 2.

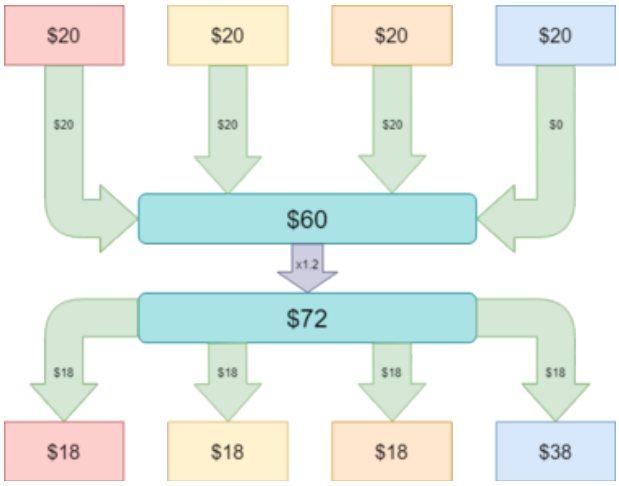
- Jogo dos bens-públicos

O jogo dos bens-públicos ou *public goods*, como é conhecido em vários lugares do mundo, também é um jogo bastante utilizado para estudos na área econômica, consiste em um grupo de jogadores em que cada um possui seus próprios bens privados. Cada um desses jogadores investe a quantia que deseja e da forma que bem entender, em um "órgão" público, uma bolsa de rendimentos ou algo do tipo. Vale lembrar que o valor investido por cada um é desconhecido para os demais jogadores.

Após arrecadado o investimento de cada um dos jogadores, esses valores são somados e multiplicados por um fator que representa basicamente o rendimento, o motivo do investimento. Ao fim, o valor total, somado com o rendimento, é dividido de forma igualitária entre todos os investidores, podendo resultar em ganhos ou perdas, dependendo do valor investido por cada um.

Assim como o Jogo do Ultimato, ele deixa claro a questão da colaboração: quanto mais os jogadores investirem, maior será o rendimento no geral, resultando em um maior ganho para todos. Por outro lado, quando um jogador não realiza um investimento o próprio tem um ganho maior, já que receberá o mesmo retorno dos demais, como mostrado na Figura 9:

Figura 9 - Representação do Jogo dos bens-públicos



Fonte: Wikipedia.[18]

[18] Disponível em: en.wikipedia.org/wiki/Public_goods_game. Acesso em: 08 nov. 2019.

APÊNDICE B - CAPTURAS DE TELA DO JOGO CORONEL BLOTTO

Aqui são apresentadas algumas capturas de tela da execução do jogo, exibindo alguns fluxos de execução.

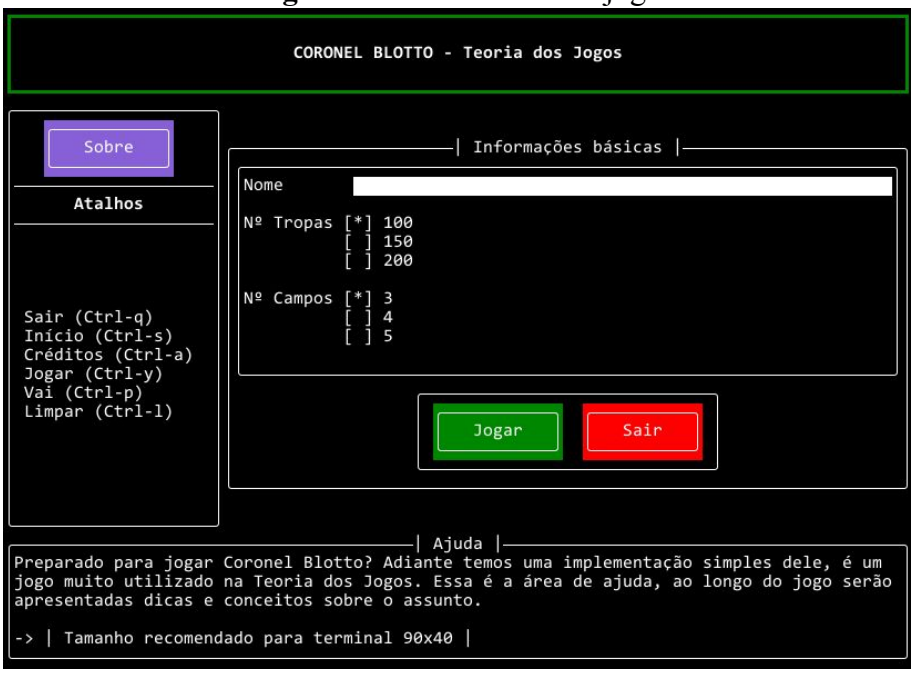
A execução do jogo inicia na tela inicial, mostrada na Figura 10, onde o jogador pode colocar o seu nome para o identificá-lo, é possível também configurar a quantidade de tropas e campos de batalha, bem como também sair ou navegar até a tela de créditos. No decorrer da execução o jogador pode utilizar as teclas de atalhos que estão dispostas no painel esquerdo.

Na tela de créditos (Figura 11) é possível navegar entre links que abordam o assunto de uma forma geral, como também conhecer outros jogos trabalhados na Teoria dos Jogos.

Na tela de distribuição das tropas o jogador deve escolher a quantidade de tropas para cada campo de batalha, lembrando-se do limite de tropas. No painel da direita é atualizado a quantidade de tropas que ainda estão disponíveis, ajudando na distribuição, conforme mostrado na Figura 12.

Após distribuir as tropas à sua maneira, o jogador pode prosseguir para a tela de resultado (Figura 13), onde é apresentada a situação em cada campo de batalha. Também é exibido algum conceito da Teoria dos Jogos na parte inferior.

Figura 10 - Tela inicial do jogo

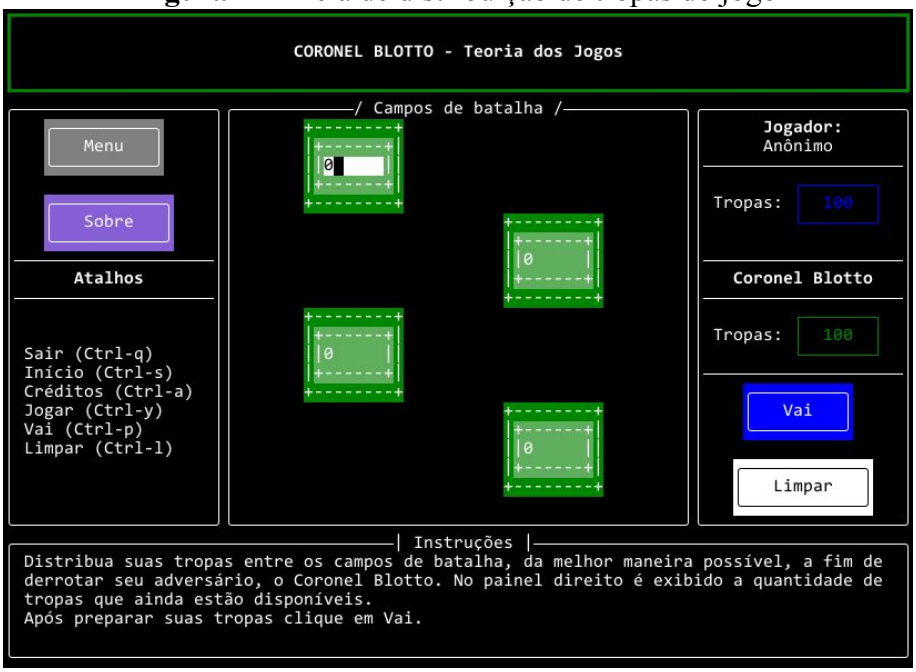


Fonte: Elaborada pelo autor.

Figura 11 - Tela de créditos do jogo

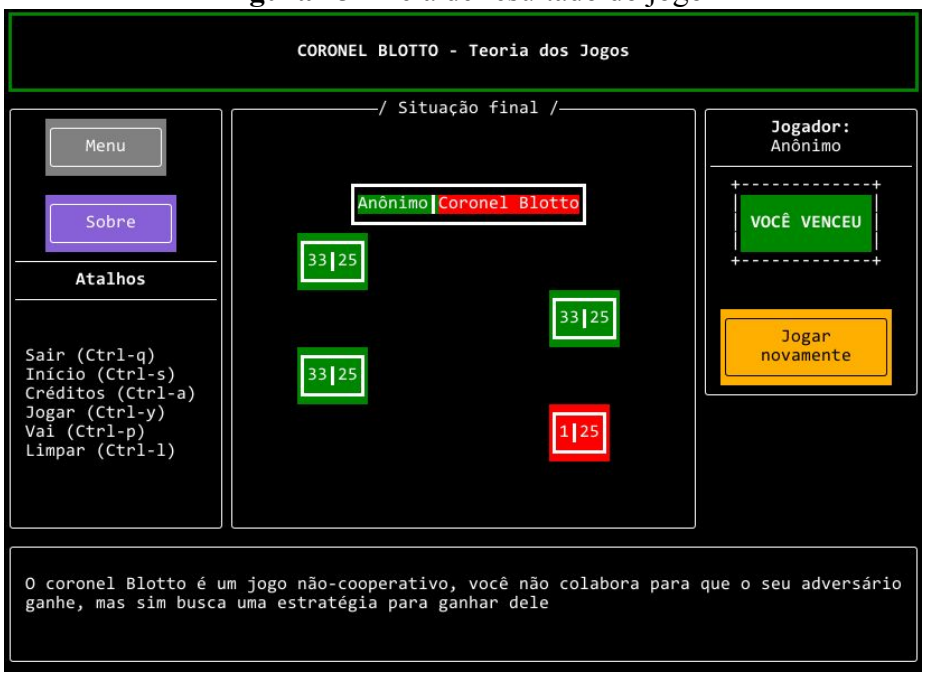
Fonte: Elaborada pelo autor.

Figura 12 - Tela de distribuição de tropas do jogo



Fonte: Elaborada pelo autor.

Figura 13 - Tela de resultado do jogo



Fonte: Elaborada pelo autor.