UNIVERSIDADE FEDERAL DE SANTA MARIA
CENTRO DE TECNOLOGIA
CURSO DE GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

René Gargano Ferrari

UMA COMPARAÇÃO ENTRE ABORDAGENS DE IA PARA O JOGO RISK

Santa Maria, RS
2022

René Gargano Ferrari

UMA COMPARAÇÃO ENTRE ABORDAGENS DE IA PARA O JOGO RISK

Trabalho de Conclusão de Curso apresentado ao Curso de Graduação em Ciência da Computação, Área de Concentração em Ciências Exatas e da Terra, da Universidade Federal de Santa Maria (UFSM, RS), como requisito parcial para obtenção do grau de **Bacharel em Ciência da Computação**.

ORIENTADOR: Prof. Joaquim Vinicius Carvalho Assunção

510
Santa Maria, RS
2022

René Gargano Ferrari

UMA COMPARAÇÃO ENTRE ABORDAGENS DE IA PARA O JOGO RISK

Trabalho de Conclusão de Curso apresentado ao Curso de Graduação em Ciência da Computação, Área de Concentração em Ciências Exatas e da Terra, da Universidade Federal de Santa Maria (UFSM, RS), como requisito parcial para obtenção do grau de **Bacharel em Ciência da Computação**.

Aprovado em 17 de fevereiro de 2022:

Joaquim Vinicius Carvalho Assunção, Dr. (UFSM)
(Presidente/Orientador)

Cesar Tadeu Pozzer, Dr. (UFSM)

Sérgio Luis Sardi Mergen, Dr. (UFSM)

Santa Maria, RS
2022

RESUMO

UMA COMPARAÇÃO ENTRE ABORDAGENS DE IA PARA O JOGO RISK

AUTOR: René Gargano Ferrari
ORIENTADOR: Joaquim Vinicius Carvalho Assunção

Ao longo das últimas décadas, jogos têm se provado ótimos ambientes para testes na área de inteligência artificial por possuírem regras bem definidas e métodos claros de avaliação. Por isso, esta monografia propõe o desenvolvimento, análise e comparação de agentes para o jogo Risk, um famoso jogo de estratégia de tabuleiro baseado em turnos, visando assim, o avanço na área de inteligência artificial. Para isso, os seguintes métodos serão propostos para a construção de quatro agentes: heurística e busca em árvore Monte Carlo. Os agentes irão se enfrentar em partidas de dois jogadores, documentando-se a porcentagem de vitórias dos *bots* contra cada um de seus adversários e o número médio de tropas pertencentes a cada agente no final do jogo. A expectativa é que sejam identificadas as vantagens, desvantagens e a eficiência de cada modelo de treino, chegando assim à conclusão de qual agente tem melhor desempenho.

Palavras-chave: Risk. Jogos de tabuleiro. Inteligência artificial. Agentes. Aprendizado de máquina. Monte Carlo.

ABSTRACT

A COMPARISON BETWEEN AI APPROACHES FOR RISK AGENTS

AUTHOR: René Gargano Ferrari
ADVISOR: Joaquim Vinicius Carvalho Assunção

Over the last few decades, games have proven to be great test environments in the artificial intelligence field due to its well defined rules and clear evaluation methods. Therefore, this monography proposes the development, analysis and comparison of agents for the game Risk, a famous strategy board game based on turns, aiming the advance in the artificial intelligence field. For that, the following methods will be proposed to the agents development: heuristic and Monte Carlo tree search. The agents are going to face each other in two player matches, documenting the bots' win rate against each of their opponents and the average number of troops owned by each agent at the end of the game. The expectation is to identify the pros, cons and efficiency of each trained model, concluding at the end which agent has the best performance.

Keywords: Risk. Board games. Artificial intelligence. Agents. Machine learning. Monte Carlo.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

RL	Reinforcement Learning
IA	Inteligência Artificial
TD	Temporal Difference
FR	Fator de Ramificação
$FMAR$	Fator Máximo de Ramificação
FMR	Fator Médio de Ramificação
$BAMC$	Busca em Árvore Monte Carlo

SUMÁRIO

1 INTRODUÇÃO

A área da Inteligência Artificial (IA) vem ganhando força nas últimas décadas, crescendo muito e acumulando participações cada vez mais importantes em pesquisas e tomadas de decisões no mundo. Um indicativo disso é o número de artigos lançados anualmente sobre esse assunto, que vem crescendo de forma significativa nos últimos anos (DUAN; EDWARDS; DWIVEDI, 2019), assim como o seu uso aplicado em diversas áreas.

Pensando nisso, já há alguns anos, vem-se estudando o uso de jogos como campo para pesquisa de IA, pois possuem regras bem definidas e métodos claros de avaliação. Algoritmos desenvolvidos para problemas amostra podem ser reutilizados para solucionar problemas similares na vida real. Por esses motivos, é válido utilizar-se de jogos para ensinar comportamentos humanos à máquina. Algumas abordagens para agentes, inclusive, já atingiram níveis extremamente satisfatórios. Tratando-se de jogos de tabuleiro pode-se citar o algoritmo desenvolvido pela Deepmind para Go (SILVER et al., 2016) e a *engine* de código aberto Stockfish para Xadrez (COSTALBA; KIISKI; ROMSTAD, 2008). Já em jogos digitais, tem-se alguns exemplos mais modernos como o Alphastar (VINYALS et al., 2019), treinado para o jogo Starcraft, e o OpenAI Five (BERNER et al., 2019), utilizado para jogar DotA 2 contra cinco jogadores. Ambas as IAs foram desenvolvidas também pela Deepmind.

Seguindo nesse âmbito, quando fala-se de agentes para jogos capazes de enfrentar humanos, existem duas situações a serem consideradas: os jogos que exigem apenas tomadas de decisões por parte do jogador, como por exemplo Xadrez, Pôquer e Truco, e os jogos que, além de tomadas de decisões, também exigem uma agilidade mental e motora, como o DotA 2, Starcraft e Counter-Strike: Global Offensive. No segundo caso, há uma preocupação sobre as regras se manterem justas, pois o ser humano sai na desvantagem por não conseguir perceber elementos na tela, tomar decisões e realizar ações tão rápido quanto a máquina. Por esse motivo, há uma complexidade maior em se construir uma IA, já que é preciso medir se o bom desempenho do agente é realmente fruto das boas decisões que ele está tomando, ou se simplesmente é um efeito da vantagem natural que ele possui.

Tendo em mente o que foi citado até aqui, mostra-se atrativo o desenvolvimento de agentes que se aproximem ou até superem as habilidades de um ser humano através de treino, observação e outros possíveis métodos. Por isso, o jogo Risk (conhecido também pela sua versão brasileira feita pela *Grow* chamada War), por ser um jogo de estratégia mundialmente famoso e que exige do jogador apenas a tomada de decisões, mostra-se um ambiente extremamente favorável para o desenvolvimento desta pesquisa. Alguns poucos trabalhos envolvendo a construção de agentes para o jogo já foram feitos (OLSSON, 2005; WOLF, 2005; CARR, 2020; GIBSON; DESAI; ZHAO, 2010), assim como análises de pos-

síveis estratégias, de batalha, ótimas para o jogo (GEORGIOU, 2004), porém, ao contrário de outros jogos, ainda não há um claro detentor do estado da arte.

Este trabalho propõe o desenvolvimento de novos agentes baseados em diferentes técnicas de IA com o objetivo de compará-los e obter *insights* sobre as melhores práticas. Para que isso seja feito, será utilizada uma implementação de uma forma simplificada do jogo, a qual permite a competição entre agentes e a análise de seus desempenhos.

Este modelo de implementação utilizado tem o objetivo de não só ser utilizado no âmbito da pesquisa, mas também como uma ferramenta para educação. A implementação do jogo é de simples utilização e alta usabilidade, permitindo assim que possam ser criados agentes em qualquer linguagem capaz de editar arquivos de texto. Dessa forma, futuramente, até estudantes que estão iniciando a sua jornada na área de IA poderão desenvolver um *bot* para o jogo, aumentando assim o número de testes realizados neste assunto e, também, facilitando o aprendizado destes estudantes.

1.1 DEFINIÇÃO DO PROBLEMA

Risk é um jogo de tabuleiro muito famoso no mundo inteiro, porém ainda não possui um estado da arte quando se trata de agentes capazes de jogá-lo. Qual o melhor modelo de agente para jogar o jogo Risk? Quais as vantagens, desvantagens e eficiência de cada modelo de treino? Este trabalho tem o objetivo de dar um passo na direção de respostas a esses questionamentos.

1.2 OBJETIVO

1.2.1 Objetivo Geral

Desenvolvimento, comparação e avaliação de modelos de agentes capazes de jogar de forma eficiente uma versão simplificada do jogo de tabuleiro Risk.

1.2.2 Objetivos Específicos

- Desenvolvimento de agentes de treino para o jogo Risk.
- Comparar execuções e obter métricas para todos os agentes desenvolvidos.

1.3 ORGANIZAÇÃO DESTA MONOGRAFIA

No próximo capítulo, a fundamentação teórica necessária para o desenvolvimento do artigo é explicada. No capítulo 3 a ambientação é explicada, assim como a implementação do jogo em Python e o funcionamento da sua comunicação com agentes externos. No capítulo 4 é abordada a implementação dos agentes. No capítulo 5 são mostrados os testes realizados e seus resultados. Finalmente, no último capítulo desenvolve-se a conclusão final em que se chegou a partir do estudo realizado, elencando também algumas ideias para trabalhos futuros que podem ser implementadas para continuar e melhorar este trabalho.

2 FUNDAMENTAÇÃO TEÓRICA

2.1 TRABALHOS RELACIONADOS

Alguns trabalhos envolvendo abordagens de IA para Risk já foram realizados. Apesar deles não chegarem a um estado da arte, várias evoluções foram feitas nas técnicas utilizadas para os agentes. Além disso, a ideia de implementação do jogo, onde os agentes se comunicam através de arquivos de texto, já foi testada em artigo com o jogo 7 Wonders. Esses trabalhos serão descritos nessa seção.

Lux Delux (SILLYSOFT, 2002) é uma implementação de Risk feita em Java pela empresa Sillysoft. O jogo já contém diversas implementações de IAs com diversas estratégias das mais simples às mais complexas. O código dos agentes é *open source*, permitindo que o usuário modifique e crie novas IAs para o jogo.

F. Olsson (OLSSON, 2005) demonstrou que é possível jogar Risk distribuindo as resoluções de problemas em um sistema de multi-agentes ou, como ele o chama, MARS (Multi-Agent Risk System). Neste sistema, um agente é distribuído em cada país do jogo enquanto um agente central fica responsável pela comunicação entre todos. Um mecanismo de votação é utilizado para a negociação entre todos os agentes do jogo para que a próxima ação seja decidida. As abordagens, no entanto, se apoiam em decisões heurísticas.

M. Wolf (WOLF, 2005) apresenta uma análise da complexidade do jogo Risk. Ele analisou a árvore de possibilidades do jogo. Os dados mostraram que a árvore do Risk, dependendo do número de tropas em campo, pode ultrapassar as árvores dos jogos Xadrez e Go, sugerindo que a estratégia de usar puramente árvores de decisão não é algo praticável no Risk. Além disso, é feita uma implementação de uma IA básica utilizando-se uma função de avaliação linear com métodos heurísticos. Após elencados os defeitos desse agente, um novo agente mais complexo é apresentado, utilizando objetivos e planos de alto nível para suprir os defeitos do agente antigo. Para que o agente aprimorado atinja pesos ótimos na função de avaliação, *temporal difference* (TD) *learning* é utilizado. Apesar do aumento na capacidade do *bot* ser apenas moderado, o autor diz que TD Learning parece ser uma técnica promissora para ser utilizada em futuros trabalhos envolvendo o jogo Risk.

J. Carr (CARR, 2020) implementou um agente para Risk utilizando TD *learning* pra treinar uma *deep neural network* utilizando uma rede de grafos convolucional para avaliar as posições dos jogadores. Esse método foi usado para buscar movimentos ótimos em uma árvore de decisões e, de acordo com Carr, necessita de pouco uso de heurística na construção da IA. Dentre o total de partidas realizadas com cinco dos melhores agentes do

Lux Delux e o agente proposto, o agente obteve 35% das vitórias, sendo assim o agente com maior número de vitórias durante os testes.

No trabalho *An implementation of the 7 Wonders board game for AI-based players* (JARDIM et al., 2020), uma implementação do jogo 7 Wonders que possibilita a interação entre IAs através de arquivos de texto é proposta. A implementação é capaz de executar centenas de partidas a cada minuto lendo arquivos json. É dito no artigo que, apesar de haver um sacrifício significativo de performance para ler e escrever os arquivos durante a execução, há um ganho na facilidade e simplicidade que se ganha no desenvolvimento de IAs para o jogo. A implementação do jogo Risk utilizada neste artigo segue esta mesma proposta, porém, ao contrário de 7 Wonders, Risk não possui visão parcial e tem uma jogabilidade baseada majoritariamente em estratégia ao invés de tática.

T. Pavin (PAVIN, 2022) fala sobre o desenvolvimento da implementação do jogo Risk utilizada nesta monografia. A implementação foi feita inteiramente em Python e permite o desenvolvimento de agentes em qualquer linguagem que consiga ler e escrever em arquivos de texto. O objetivo da implementação é facilitar o desenvolvimento de agentes para que seja de fácil utilização por estudantes que estão iniciando na área de Inteligência Artificial. A troca de informações entre o jogo e os agentes desenvolvidos é feita através de arquivos json. Da mesma forma que na implementação do jogo 7 Wonders, há uma perda de performance na execução do jogo em troca da simplicidade adquirida no desenvolvimento das IAs.

2.2 VISÃO GERAL

Risk é um jogo de estratégia de dois a seis jogadores criado pelo francês Albert Lamorisse em 1957. Originalmente era chamado *La Conquete du Monde*, porém, em 1959 foi adaptado e publicado pelos *Parker Brothers* com o nome de *Risk: The Game of Global Domination* nos Estados Unidos. A figura 2.1 ilustra o tabuleiro do jogo. Como existem diferentes versões do jogo produzidas ao longo dos anos, as regras possuem algumas variações dependendo de qual versão é utilizada. Desde 1991 os direitos do jogo são de posse da *Hasbro*. Além disso, o jogo foi adaptado e lançado no Brasil com o nome de *War* pela *Grow* em 1971.

2.3 REGRAS

Existem diversas regras para o jogo dependendo da versão que é jogada, afinal o jogo foi sendo relançado e atualizado para diversos países com o passar dos anos. As

Figura 2.1 – Tabuleiro do Jogo

Fonte: https://www.ultraboardgames.com/risk/game-rules.php

regras utilizadas nesta monografia são as regras básicas originais do jogo lançado nos Estados Unidos, porém sem o uso de cartas por motivo de simplificar a implementação.

2.3.1 Objetivo do Jogo

De acordo com o livro de regras do jogo (HASBRO, 1959), o objetivo principal é: “Conquistar o mundo ocupando todos os países no tabuleiro. Você precisa eliminar todos os seus oponentes.”

2.3.2 Configuração Inicial do Jogo

Nesta monografia, serão realizadas partidas apenas entre dois jogadores. Nesse caso, cada jogador recebe 40 tropas e 21 países aleatórios no início do jogo. As tropas dos jogadores são distribuídas aleatoriamente entre os países por eles pertencidos, sendo que cada país precisa ter no mínimo uma tropa.

2.3.3 Forma de Jogar o Jogo

Risk é um jogo de turnos. O jogo inicia com o primeiro jogador. Ele pode realizar suas ações em seu turno enquanto os outros jogadores devem esperar sua vez. Quando

todos tiverem jogado seus turnos, a vez volta para o primeiro jogador e a ordem de turnos é repetida novamente até o fim do jogo. Durante o seu turno, o jogador pode realizar as seguintes ações na respectiva ordem:

1. Mobilização
2. Ataque e Conquista
3. Fortificação

2.3.4 Mobilização

No início do turno de cada jogador, tropas são distribuídas a ele. A cada três países o jogador ganha uma tropa nova, ganhando no mínimo três tropas por turno. O número de tropas novas distribuídas obedece a fórmula 2.1. Além disso, caso o jogador possua algum continente inteiro ocupado, receberá tropas bônus referentes àquele continente. O número de tropas bônus fornecidas por cada continente pode ser visualizado na tabela 2.1.

$$\textit{Tropas Novas} = max(\frac{\textit{Países Ocupados}}{3}, 3) + \textit{Tropas Bônus dos Continentes} \quad (2.1)$$

Tabela 2.1 – Tropas Bônus para Continentes Ocupados

Continente	**Tropas Bônus**
Ásia	7
América do Norte	5
Europa	5
África	3
América do Sul	2
Austrália	2

Fonte: Autor

2.3.5 Ataque

Durante a etapa de ataque, o jogador pode utilizar qualquer país conquistado por ele, que tenha mais de uma tropa, para atacar os países inimigos que façam fronteira com

esse país. O jogador deve escolher com quantos dados irá atacar. Para cada tropa extra no país, considerando que todo país necessita de no mínimo uma tropa, é possível adicionar um dado até o máximo de três, como é possível visualizar na tabela 2.2.

Tabela 2.2 – Mínimo de tropas no país para utilizar os dados

Tropas no País	Dados
2	1
3	2
4	3

Fonte: Autor

Quando um ataque é feito, tanto o atacante quanto o defensor rolam seus dados. O defensor irá rolar um dado se tiver apenas uma tropa defendendo o país e dois dados caso tenha duas ou mais tropas. Após rolados os dados, os valores obtidos mais altos de cada jogador são comparados em ordem, ou seja, os maiores valores são comparados e em seguida os segundos maiores, caso o defensor tenha defendido com dois dados, também. Se o atacante tirar um valor maior, o defensor perde uma tropa, mas se o defensor tirar um valor maior ou igual, o atacante perde uma tropa.

2.3.6 Conquista

Quando a última tropa de um país é derrotada, o jogador atacante pode mover quantas tropas desejar do país atacante para o país recém conquistado. No mínimo uma tropa deve ser movida.

2.3.7 Fortificação

Antes de finalizar o seu turno, o jogador tem a oportunidade de fortificar uma posição se desejar. Isso é feito enviando quantas tropas desejar de um país aliado escolhido para outro, desde que ambos estejam conectados por países aliados e que o país remetente fique com no mínimo uma tropa após a mudança.

2.4 COMPLEXIDADE

Um estudo sobre a complexidade do jogo já foi feito por M. Wolf (WOLF, 2005). Alguns pontos importantes serão retomados nesta seção, pois são necessários no desen-

volvimento dos agentes.

A complexidade do jogo pode ser medida através da complexidade do espaço de estados e da complexidade da árvore do jogo. A complexidade do espaço de estados trata-se do número de estados possíveis no jogo. Já a complexidade da árvore do jogo é a medida da complexidade das decisões combinada com a duração do jogo medida em turnos. A complexidade das decisões pode ser expressa através do fator de ramificação. No Risk, cada turno é composto por várias etapas, as quais exigem diferentes tomadas de decisões e não tem um número fixo de ações a serem executadas.

2.4.1 Complexidade do Espaço de Estados

No jogo Risk, o número de estados depende do número de tropas presentes no tabuleiro. Isso acontece pois todas as ações do jogo envolvem tropas e podem ser feitas de formas diferentes. Pegando a ação de ataque como exemplo, o jogador terá a opção de atacar com um, dois ou três dados, considerando que o país atacante tem quatro ou mais tropas, ou simplesmente não atacar. O jogador poderá atacar quantas vezes quiser até que o país inimigo seja tomado ou até que reste apenas uma tropa no país aliado. Isso significa que o número de possíveis ataques depende tanto do número de tropas no país aliado quanto do número de tropas no país inimigo. Como o limite de tropas em cada país e no tabuleiro é infinito, o número de possíveis estados durante a partida também é infinito. Ainda assim, é possível calcular o espaço de estados dado um número máximo de tropas aceito no tabuleiro através da fórmula 2.2.

$$\textit{Estados do Jogo}(M, J) = \textit{Distribuição das Tropas} \times \textit{Distribuição dos Países} \tag{2.2}$$

$$= \sum_{I=T}^{M} \binom{T + (I-T) - 1}{(I-T)} \times \binom{J + T - 1}{T} \tag{2.3}$$

$$= \sum_{I=T}^{M} \binom{I-1}{I-T} \times \binom{J + T - 1}{T} \tag{2.4}$$

Sendo *M* o número máximo de tropas aceito no tabuleiro;
J o número de jogadores e
T o número total de países no mapa.

Sendo assim, é possível medir, através da observação de uma quantidade grande de partidas, o número médio de tropas presentes em uma partida de Risk e utilizá-lo para calcular o espaço de estados médio de uma partida. Já tratando-se do jogo físico de tabuleiro, de acordo com M. Wolf (WOLF, 2005), as várias versões de Risk geralmente

suportam aproximadamente um máximo de 200 tropas por jogador, então, como exemplo, para calcularmos a complexidade do jogo físico para quatro jogadores, usaremos $M = 800$, como demonstrado na equação 2.5.

$$\textit{Estados do Jogo}(800, 4) = \sum_{I=42}^{800} \binom{I-1}{I-42} \times \binom{45}{42} \tag{2.5}$$

$$= \sum_{I=42}^{800} \frac{(I-1)!}{((I-1)-(I-42))! \times (I-42)!} \times \frac{45!}{(45-42)! \times 42!} \tag{2.6}$$

$$= \sum_{I=42}^{800} \frac{(I-1)!}{(42-1)! - (I-42)!} \times \frac{45!}{3! \times 42!} \tag{2.7}$$

$$= 3.029 \times 10^4 3 \times 14,190 \tag{2.8}$$

$$= 4.298 \times 10^{47} \tag{2.9}$$

$$\approx 10^{47} \tag{2.10}$$

2.4.2 Fator de Ramificação

O fator de ramificação mede a complexidade das decisões do jogo. Ele envolve a frequência com que as decisões são tomadas a cada turno e o número de ações válidas a serem tomadas. O fator de ramificação (FR) de um estado do jogo é o número de todos os estados possíveis de se chegarem a partir do estado atual.

- **Fator Máximo de Ramificação**

  O fator máximo de ramificação (FMAR) do jogo é o maior fator de ramificação de todos os estados. Como no Risk o número de ações em um estado pode depender do número de tropas e o limite de tropas no jogo é infinito, logo o FMAR é infinito também.

- **Fator Médio de Ramificação**

  O fator médio de ramificação (FMR) do jogo é a média aritmética do FR de cada estado do jogo. Como o FMAR é infinito e não há um número fixo de estados por jogo, o FMR não terá um valor exato. Porém, como ele é importante para obter-se a complexidade da árvore do jogo, é possível estimá-lo calculando a média aritmética da aproximação de todos os FR do jogo. Essa aproximação foi calculada por M. Wolf (WOLF, 2005) medindo a frequência que cada decisão é tomada e o número

de ações válidas por ocorrência de cada decisão. O FMR é demonstrado na fórmula 2.11.

$$FMR = MA_{t \in T}(FR_t) \tag{2.11}$$

Sendo *MA* a função que retorna a média aritmética dos dados e
T o espaço com todos os turnos medidos.

- **Decisão Média**

  Outra forma de calcular o FMR é através do cálculo da média da frequência das decisões tomadas a cada jogo e a média do número de ações válidas por ocorrência de decisão, como na fórmula 2.12.

$$FMR = \prod_{D \in TD} MA_{d \in D}(\textit{opções}(d))^{\textit{frequência}(D)} \tag{2.12}$$

  Sendo *TD* o conjunto com todos os tipos de decisão do jogo;
  opções(d) a função que retorna o número de estados válidos alcançáveis a partir do estado atual escolhendo uma ação válida de *d*;
  frequência(D) a função que retorna a ocorrência por turno de *D* e
  MA a função que retorna a média aritmética dos dados.

2.4.3 Complexidade da Árvore do Jogo

A árvore do jogo é uma árvore onde cada nó é um estado do jogo e os seus nós filhos são os possíveis estados que podem ser atingidos a partir do estado atual através de diferentes ações. A raiz é o estado inicial do jogo. Jogos sem uma limitação de ações têm a árvore do jogo infinita. No caso do Risk, como suas ações dependem do número e disposição das tropas no tabuleiro, sendo que não há uma limitação de tropas em campo, sua árvore acaba sendo infinita.

Ainda assim, é possível calcular a complexidade média da árvore do jogo (CMA) elevando-se o fator médio de ramificação (FMR) ao número médio de turnos por jogo (MTJ) 2.13. Pode-se notar que na tabela 2.3 o jogo Risk possui duas complexidades médias de árvore. Isso acontece pois foram sugeridas por M. Wolf (WOLF, 2005) duas fórmulas distintas para o cálculo do FMR, podendo-se optar por uma delas.

$$CMA = FMR^{MTJ} \tag{2.13}$$

2.4.4 Comparando Risk com Outros Jogos

Após calcularmos a complexidade do espaço de estados e a complexidade média da árvore do jogo, comparando-as com as de outros jogos clássicos de tabuleiro, logo notamos que Risk apresenta uma complexidade maior. Isso significa que métodos de busca em árvore muito utilizados para a construção de agentes em jogos como o Minimax não podem ser aplicados ao Risk, pois não são escaláveis para árvores com alto FR. A comparação pode ser vista na tabela 2.3.

Além disso, a complexidade de estados do Risk deve-se ao número de tropas em campo, o que faz com que a diferença entre os estados seja muito mais sutil do que nos demais outros jogos mostrados na tabela. O impacto dos estados, em algumas situações, se mostra menor conforme o número total das tropas aumenta. Um exemplo disso é a diferença grande entre dois estados onde, em um deles, um país possui duas tropas em seu território e, no outro, possui três. No primeiro estado, o país pode atacar com apenas um dado, enquanto que no segundo, pode atacar com duas. Já quando a diferença entre os estados se dá por em um deles o país ter 30 tropas e no outro 31, o impacto que essa diferença surtirá será menor. Afinal, se um país com 31 tropas perde apenas uma tropa, ele continua com um número significante de tropas em seu território, oferecendo ainda a mesma periculosidade aos territórios inimigos.

Tabela 2.3 – Complexidades de Jogos de Tabuleiro Clássicos

Jogo	**Complexidade do Espaço de Estados**	**Complexidade Média da Árvore de Jogo**
Damas	10^{31}	10^{18}
Xadrez	10^{46}	10^{123}
Risk (200 tropas)	10^{47}	$10^{2350} \mid 10^{5945}$
Risk (1000 tropas)	10^{78}	$10^{2350} \mid 10^{5945}$
Go	10^{172}	10^{360}
Risk	∞	$10^{2350} \mid 10^{5945}$

Fonte: (WOLF, 2005)

2.5 DEFINIÇÃO DE AGENTE

Alguns agentes serão desenvolvidos nesta monografia. Em IA, um agente inteligente é qualquer coisa que percebe o ambiente a sua volta e toma ações com base nisso. O agente pode aprender com o passar do tempo ou simplesmente ter o seu comportamento pré-definido.

O livro *Artificial Intelligence: A Modern Approach* (RUSSELL; NORVIG, 2009) define um agente como: “Qualquer coisa que perceba seu ambiente por meio de sensores e aja sobre esse ambiente por meio de atuadores”, define um agente racional como: “Um agente que atua de forma a maximizar o valor esperado de uma medida de desempenho com base na experiência e conhecimento anteriores” e define o campo de inteligência artificial como: “O estudo e projeto de agentes racionais”.

2.6 BUSCA EM ÁRVORE MONTE CARLO

A árvore do jogo conecta possíveis estados do jogo a serem alcançados através de ações. A partir de cada estado é possível escolher uma das ações disponíveis para ser levado a um novo estado (nodo). Existem algoritmos simples para se fazer busca em árvore, como a busca em largura e a busca em profundidade, que seguem uma determinada ordem para visitar cada nodo da árvore, porém eles não têm uma escalabilidade tão boa quando se trata de árvores grandes. Por isso existem algoritmos mais inteligentes, que buscam atribuir valores aos estados, para que assim possam aprender ao longo das execuções o melhor caminho a se percorrer. A isso dá-se o nome de *Reinforcement Learning* (RL).

RL é uma área de *Machine Learning* que visa aprender boas estratégias para solucionar um certo problema. Um jeito de fazer isso é repetidamente usar as melhores ações encontradas para o problema, porém há sempre uma chance de que a ação escolhida não seja a mais otimizada. Para isso, continuamente novos estados são avaliados durante a etapa de aprendizado para que se encontrem novas ações ótimas. Isso é conhecido em RL como etapas de *exploration* e *exploitation*. O valor de um estado é armazenado no próprio estado, sendo atualizado sempre que o algoritmo é executado novamente.

A busca em árvore Monte Carlo (BAMC) funciona dessa forma, repetindo as etapas de *exploitation* e *exploration* na árvore do jogo até que se conheça um caminho próximo do ótimo. Cada rodada da execução da BAMC, de acordo com C. Browne (BROWNE et al., 2012), consiste nas seguintes etapas:

- Selection: seleciona um caminho já conhecido a partir do estado atual R (raiz) até alguma folha F.
- Expansion: caso F não finalize o jogo, expande a árvore e seleciona algum estado filho C ainda não visitado para explorar a árvore.
- Simulation: simula o restante do jogo a partir do estado não explorado C, retornando se houve vitória ou derrota.

- Backpropagation: usa o resultado da simulação para atualizar os valores dos estados entre C e R.

A etapa de *selection* é encarregada do *exploitation*, onde segue-se um caminho já conhecido, que teve até o dado momento o melhor desempenho. Já durante as etapas de *expansion* e *simulation*, é feito o processo de *exploration*, onde nodos nunca antes visitados são testados para serem avaliados.

Durante a etapa de simulação podem-se aplicar duas estratégias: o *light rollout*, que consiste em tomar ações aleatórias até o término do jogo, e o *heavy rollout*, o qual toma ações baseado em uma heurística. No modelo implementado nesta monografia foi aplicado o *heavy rollout*.

A principal dificuldade em percorrer uma árvore utilizando BAMC é manter um balanço entre as etapas de *exploitation* e *exploration*. Uma fórmula muito utilizada para balanceamento dessas etapas é chamada de *Upper Confidence Bound applied to trees* (UCT), demonstrada por C. Browne (BROWNE et al., 2012). Ela pode ser visualizada na equação 2.14.

$$UCT = \frac{w_i}{n_i} + c\sqrt{\frac{\ln N_i}{n_i}} \tag{2.14}$$

Sendo w_i o saldo de vitórias e derrodas do nodo;
n_i o número de simulações que o nodo participou;
N_i o número total de simulações que o pai do nodo participou;
c o parâmetro de exploração, teoricamente equivalente a $\sqrt{2}$, mas que também pode ser escolhido empiricamente.

Essa fórmula é utilizada para tomar a decisão de que nó seguir na etapa de *exploitation*. Dessa forma, a escolha é tomada não só com base no valor do nodo (saldo de vitórias e derrotas), mas também com base no número de visitas que o nodo e seu pai já tiveram, permitindo assim que nodos pouco visitados também possam ser melhor explorados.

A função de *backpropagation* da BAMC acontece percorrendo o caminho feito durante a iteração de volta até a raiz, incrementando o contador de iterações de cada estado e aumentando ou diminuindo os seus valores dependendo do resultado da simulação (se ganhou ou perdeu). No caso de dois jogadores com rodadas alternadas um aprimoramento pode ser feito, diminuindo o valor da recompensa conforme se chega próximo a raiz. Este aprimoramento é chamado de *BackupNegaMax* (BROWNE et al., 2012). Esse aprimoramento é análogo à variante da busca Minimax chamada Negamax.

Um algoritmo muito similar à BAMC e também muito usado em jogos é o Minimax, que também atribui valores aos nodos, percorrendo a árvore inteira atrás do melhor caminho. O problema é que ele não é escalável para árvores com alto fator de ramificação,

já que precisa avaliar cada possível estado para avaliar qual o melhor. Existem alguns aprimoramentos para melhorar sua eficiência em casos de fator de ramificação alto, como a poda alpha-beta, porém, mesmo assim, nesses casos a BAMC se destaca, pois não precisa percorrer todos os possíveis estados, focando apenas nas sub-árvores consideradas mais relevantes.

Apesar de, teoricamente, o método de avaliação dos estados usado na BAMC convergir para o Minimax, como lembrado por C. Browne (BROWNE et al., 2012), a versão básica da BAMC converge apenas em jogos chamados “Monte Carlo Perfeitos”, que são jogos onde é possível obter uma jogada ótima para cada estado visitado caso sejam explorados cada um de seus filhos. O Risk não é um jogo “Monte Carlo Perfeito”, pois tem uma complexidade de espaço de estados infinita, tornando impossível a exploração de todos os nodos. Apesar disso, a BAMC apresenta várias vantagens sobre a poda alpha-beta e outros algoritmos similares que minimizam o espaço de busca na árvore.

Uma das principais vantagens é que a BAMC pura não precisa de uma função de avaliação explícita. Simplesmente implementar as mecânicas do jogo é o suficiente para explorar o espaço do jogo. Isso ocorre porque cada estado do jogo guarda através dos seus filhos já visitados um histórico de melhores e piores ações, podendo assim embasar sua decisão de qual ação tomar em acontecimentos prévios. Por isso, a BAMC pode ser implementada em jogos que ainda não tem uma estratégia desenvolvida.

Além disso, a BAMC cresce de forma assimétrica, já que o método se concentra nas sub-árvores mais promissoras. Por isso ela alcança melhores resultados que outros algoritmos clássicos usados em jogos que possuem um fator de ramificação alto.

Já um lado ruim é que algumas jogadas que parecem fortes de início podem acabar levando a uma derrota, enquanto que algumas jogadas fracas podem eventualmente levar a jogadas mais fortes. Infelizmente o algoritmo pode acabar não percebendo essas jogadas devido a poda. Ele não irá visitar mais esse estado aparentemente fraco na etapa de *exploitation* devido o seu baixo valor e não o visitará na etapa de *exploration*, pois ele já foi visitado.

3 ARQUITETURA E IMPLEMENTAÇÃO DO RISK

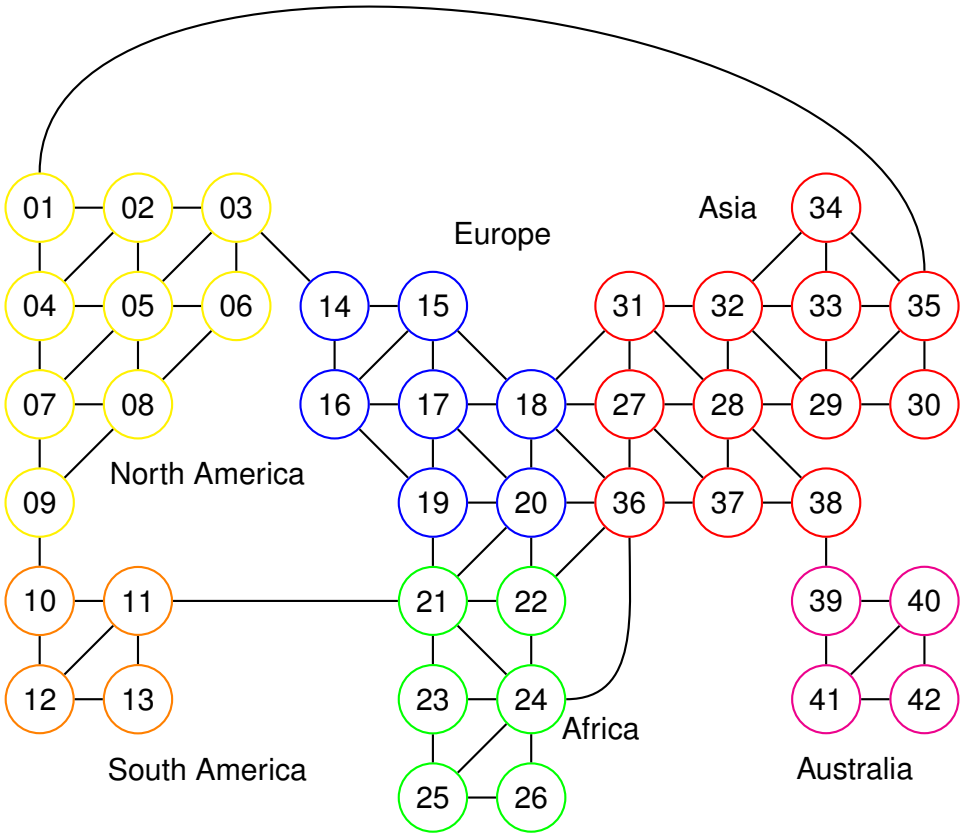
3.1 PROPÓSITO

A implementação utilizada nesta monografia foi criada com o objetivo de facilitar a criação de agentes para Risk. A razão disso é fazer com que o trabalho tenha um viés tanto científico quanto educativo. Os agentes criados aqui podem ser facilmente reproduzidos em qualquer outra linguagem que consiga realizar escrita e leitura de arquivos, fazendo assim com que o trabalho possua um alto grau de replicabilidade e, além disso, continuidade, já que novos agentes também podem ser facilmente criados. Nesta monografia, foca-se na explicação da implementação dos agentes e da comunicação deles com o jogo. Uma explicação mais detalhada sobre a implementação do jogo pode ser encontrada na monografia feita por T. Pavin (PAVIN, 2022). O código fonte do jogo pode ser encontrado no repositório *Risk-Implementation* (Pavin, Thiago and Gargano Ferrari, René, 2022).

3.2 AMBIENTE DO JOGO

O jogo é representado através de um grafo de conexão entre países/regiões. É possível visualizar uma representação do grafo do jogo na figura 3.1.

Figura 3.1 – Grafo de Conexão de Países



01 Alaska. 02 Northwest America. 03 Greenland. 04 Alberta. 05 Ontario. 06 Quebec. 07 Western America. 08 Eastern America. 09 Central America. 10 Venezuela. 11 Brazil. 12 Peru. 13 Argentina. 14 Iceland. 15 Scandinavia. 16 Great Britain. 17 Northen Europe. 18 Ukraine. 19 Western Europe. 20 Southern Europe. 21 North Africa. 22 Egypt. 23 Congo. 24 East Africa. 25 South Africa. 26 Madagascar. 27 Afghanistan. 28 China. 29 Mongolia. 30 Japan. 31 Ural. 32 Siberia. 33 Irkutsk. 34 Yakutsk. 35 Kamchatka. 36 Midle East. 37 India. 38 Siam. 39 Indonesia. 40 New Guinea. 41 Western Australia. 42 Eastern Australia.

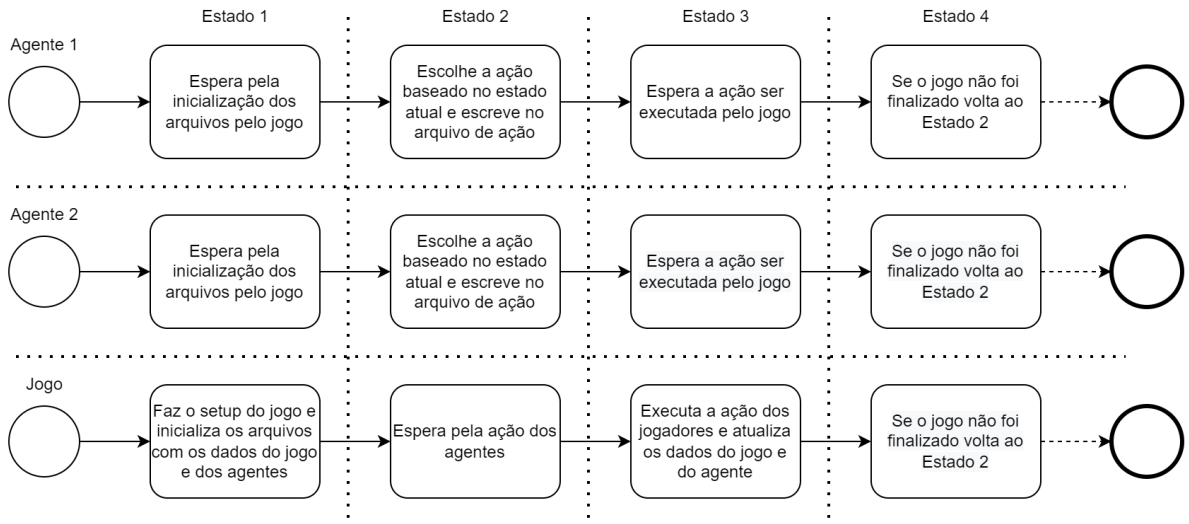
Fonte: Autor

3.3 LINHA DE EXECUÇÃO

A comunicação entre o jogo e os agentes ocorre através da troca de arquivos json. A linha de execução dos três processos é ilustrada na figura 3.2. Todos são executados simultaneamente, em processos diferentes, porém enquanto um dos processos está trabalhando, seja o jogo ou um dos agentes, os outros estão em espera ociosa aguardando a atualização dos arquivos. Os dois agentes sempre são inicializados primeiro e aguardam

a inicialização do jogo. Quando o jogo é iniciado, ele cria os arquivos json utilizados para a comunicação jogo-agente já com as informações dos estados dos agentes e do jogo. Os agentes, percebendo que os arquivos foram criados, executam a ação correspondente ao seu estado atual e, caso não estejam no estado waiting, a informam ao jogo, aguardando então a sua execução.

Figura 3.2 – Linha de Execução do Jogo



Fonte: Autor

3.4 CLASSES

- Game: É responsável pela execução do jogo, leitura e escrita dos arquivos json responsáveis pela comunicação jogo-agente e chamada dos métodos da classe *Player*.
- World: É responsável pela inicialização dos países e dos continentes através das classes *Country* e *Continent*.
- Country: Armazena informações sobre um país. São elas o seu nome, o nome de seus vizinhos, quem é seu dono e o número de tropas sobre o seu território.
- Continent: Armazena as informações sobre um continente. São elas o seu nome, os países que a ele pertencem, o número de tropas bônus fornecidas por ele e, se houver, o jogador que o possui.
- Player: Armazena informações sobre o jogador. São elas o seu id, os países aliados, o número de tropas novas disponíveis para serem colocadas em jogo, o total de

tropas aliadas dispostas sobre o tabuleiro (grafo), o estado do jogador, uma matriz de conexões entre países aliados e, por último, todos os países inimigos que fazem fronteira com os países aliados. Além disso, a classe possui métodos que podem ser chamados para que o jogador interaja com o tabuleiro. Os métodos são melhor detalhados na seção 3.6.

3.5 COMUNICAÇÃO JOGO-AGENTE

Como já dito anteriormente, o desenvolvimento dos agentes pode ser feito em qualquer linguagem capaz de realizar escrita e leitura de arquivos. Nesta monografia, foi escolhida a linguagem python para o desenvolvimento dos agentes. A comunicação jogo-agente é feita através da troca de arquivos json. O jogo fornece para cada jogador um arquivo *pn_state.json*, sendo *n* o *id* do agente, contendo os atributos atuais do jogador e o estado atual do jogo. Já o agente, precisa fornecer ao jogo outro arquivo chamado *pn_call.json* com a ação que ele deseja que seja executada. Pode-se ver a estrutura dos dois arquivos json nos pseudocódigos nos apêndices A e B.

3.6 MÉTODOS CHAMADOS PELO AGENTE

Os agentes têm a possibilidade de requisitar uma ação por turno sendo que existem quatro opções de ações no jogo. Elas estão listadas na tabela 3.1.

Tabela 3.1 – Métodos dos Agentes

Nome	**Argumentos**
attack	n_dice: int, attacker: str, attacked: str
move_troops	n_troops: int, from_country: str, to_country: str
set_new_troops	n_troops: int, country_name: str
pass_turn	

Fonte: Autor

- attack: utiliza um país aliado para atacar um país inimigo adjacente podendo escolher atacar com um a três dados. Pode ser usado no turno de ataque.
- move_troops: move tropas entre dois países aliados. Este método pode ser usado tanto na etapa de fortificação quanto de conquista de país (após matar todos as tropas de um país inimigo).

- set_new_troops: coloca tropas novas disponíveis em um país aliado de escolha no tabuleiro. Pode ser usado no turno de mobilização.
- pass_turn: Passa para o próximo turno, ou, se estiver no último turno (fortificação), passa a vez para o próximo jogador.

4 IMPLEMENTAÇÃO DOS AGENTES

4.1 METODOLOGIA DE DESENVOLVIMENTO

De início, foi criado um agente com tomada de decisões aleatória. Este agente serve como parâmetro de comparação para os outros. Como ele não possui uma estratégia definida, espera-se que os outros agentes desenvolvidos tenham sempre uma taxa de vitória superior a ele. O agente Random é mais aprofundado na seção 4.2.

Além desse agente, outros dois agentes com estratégias bem definidas foram criados. Ambos precisam conseguir jogar razoavelmente o jogo Risk. Isso significa que eles conseguem jogar o jogo de forma a visivelmente conseguir avançar em seu objetivo de ganhá-lo, porém não é cobrada complexidade em suas tomadas de decisões. Eles não precisam oferecer desafio a um jogador humano experiente, mas, se subestimados, precisam conseguir chegar à vitória.

O comportamento do primeiro agente com estratégia definida é baseado na heurística de um agente criado para o jogo Lux Delux, que é a implementeção de Risk em Java feita pela empresa Sillysoft, chamado Cluster. Esse agente adota um comportamento conservador, formando um conjunto a partir de países vizinhos que ele julga ter alto valor, defendendo-o e expandindo-o ao longo do jogo enquanto tenta conquistar novos conjuntos. A escolha de Cluster se dá por ele ser antecessor de dois outros agentes mais poderosos, Shaft o Quo. Agentes que, de acordo com a empresa, tiveram um desempenho muito bom em relação aos outros agentes desenvolvidos por ela. Dessa forma, há espaço para aprimoramento do agente em futuros experimentos. O agente Cluster é mais aprofundado na seção 4.3.

O segundo agente foi desenvolvido para ter um comportamento agressivo. Ele também é baseado na heurística de um dos agentes de Lux Delux. Chamado Angry, este agente tem como objetivo atacar o máximo possível, gastando todas as suas tropas no início do jogo para ganhá-lo de forma rápida. Ele foi escolhido pela sua heurística funcionar de forma oposta ao Cluster, focando em agressividade ao invés de defesa. Dessa forma, ele possibilita que o agente mais complexo possa ser testado em uma variedade maior de situações. O agente Angry é mais aprofundado na seção 4.4.

A partir disso, será possível treinar outro agente mais complexo, herdando e modificando os métodos do agente Cluster para que ele aprenda através de suas ações. Então, após o treinamento, é possível compará-lo com o agente original, possibilitando avaliar se houve evolução através do aprendizado.

Como o jogo Risk possui diversos estados nos quais o agente precisa assumir comportamentos diferentes, sendo eles: mobilização, ataque, conquista e fortificação, fica

difícil avaliar o motivo da vitória ou derrota caso todos os estados sejam modificados para aprender. Por esse motivo, decidiu-se focar em modificar apenas um deles: o ataque. Este estado foi escolhido por ser um estado em que se toma um grande número de decisões quando comparado aos outros dois estados, tendo assim um maior impacto em ditar a dinâmica do jogo.

O método escolhido a ser testado como agente é o método de busca em árvore Monte Carlo. A busca em árvore Monte Carlo permite a construção de uma sub-árvore através de buscas anteriores na árvore do jogo. Dessa forma, um caminho é formado pela árvore, evitando que seja necessária a busca pela árvore toda.

Para treinar o agente, várias partidas entre o Cluster e o Monte Carlo serão analisadas para que seja feita a construção de um caminho pela árvore do jogo com os nodos mais utilizados e bem sucedidos. O agente Monte Carlo é mais aprofundado na seção 4.5.

Por fim, para realizar a comparação entre os agentes, lembrando que cada partida contém sempre dois jogadores (agentes) se enfrentando, foram realizadas partidas com todas as combinações de agentes possíveis. Então, comparou-se as taxas de vitória de cada agente contra cada adversário diferente, como também o número médio de tropas possuído por cada agente ao final da partida. A comparação entre o número de tropas não é um fator determinante para a eficiência dos agentes, porém é algo a ser analisado e demonstrado para fins de debate. O código fonte dos agentes implementados, assim como os testes realizados entre eles podem ser encontrados no repositório *Risk-Agents* (Gargano Ferrari, René, 2022).

4.2 RANDOM

O Random tem a característica de ter todos os seus métodos baseados na aleatoriedade. Ele foi feito com o propósito de servir como parâmetro do mínimo aceitável como agente. Espera-se que todos os outros agentes tenham uma porcentagem de vitória alta em cima dele. Os seus métodos podem ser descritos da seguinte forma:

- Mobilização: o seu método de mobilização distribui aleatoriamente tropas entre os países aliados até que não existam mais tropas disponíveis para serem alocadas.
- Ataque: em seu método de ataque, ele faz ataques aleatórios a seus vizinhos até que não exista mais nenhum país disponível para atacar. Foi decidido gastar todas as tropas do agente na etapa de ataque para que o agente tenha uma característica agressiva apesar de sua aleatoriedade. Dessa forma o agente avança para a vitória caso o inimigo não ofereça resistência.
- Conquista: neste método é movido um número aleatório de tropas entre o país atacante e o conquistado.

- Fortificação: seu método de fortificação consiste em mobilizar um número aleatório de tropas entre dois países aleatórios conectados, sejam vizinhos ou conectados por países aliados.

4.3 CLUSTER

Este agente possui uma característica mais conservadora. Procurando sempre defender um bloco de países que julga possuir mais valor, tentando expandi-lo até que domine todos os países do tabuleiro.

- Mobilização: neste método, caso já se tenha obtido algum continente, escolhe-se o que fornece o maior bônus de tropas. Cria-se um cluster (aglomerado de países) com os países do continente mais os países aliados conectados a ele e então distribuí-se as tropas nos países com menos tropas da borda do cluster. Caso ainda não haja continente obtido, escolhe-se o continente com a maior razão de aliados sobre inimigos (*tropas aliadas* / *tropas inimigas*). Então escolhe-se o país aliado com maior número de tropas inimigas ao redor e coloca-se todas as tropas novas nele.
- Ataque: o método consiste em primeiro selecionar um cluster formado pelo continente que fornece mais tropas bônus e os países conectados a ele. Caso não possua um continente, forma um cluster com os países conectados ao país com mais tropas. Após selecionar o cluster, o agente irá seguir a seguinte linha de ação para atacar a partir dele:
  - Ataque Fácil: procura países inimigos com apenas uma tropa e os ataca.
  - Ataque Isolado: caso não tenha realizado o ataque fácil, procura países inimigos isolados no mapa, ou seja, sem nenhum país inimigo vizinho, e os ataca.
  - Ataque de Consolidação: caso não tenha realizado o ataque isolado, seleciona dentre os países inimigos aquele que tem maior razão de tropas vizinhas aliadas sobre tropas inimigas no país (*tropas aliadas* / *tropas inimigas*) e o ataca até que seja conquistado ou que os países aliados vizinhos não tenham mais tropas para atacar.
- Conquista: o método segue a seguinte linha de ações:
  - Se o país atacante tem apenas um país inimigo vizinho, move todas as tropas para o país conquistado.
  - Senão, se o país conquistado tem apenas um país inimigo vizinho, não move nenhuma tropa.

  - Senão, seleciona um continente como objetivo para ser conquistado.
  - Se o país conquistado não tiver países inimigos vizinhos dentro do continente alvo, não move nenhuma tropa.
  - Senão, se o país atacante não tiver países inimigos vizinhos dentro do continente alvo, move todas as tropas para o país conquistado.
  - Senão, compara os países inimigos vizinhos do país atacante e do conquistado que estão dentro do continente alvo. Passa todas as tropas para o país que tem o vizinho inimigo com menos tropas.

- Fortificação: similar ao método de ataque, forma um cluster a partir do continente adquirido que fornece mais tropas bônus ou a partir do país com mais tropas. Após ter o cluster selecionado, escolhe o país com mais tropas dentro do cluster (sem vizinhos inimigos) e move todas para o país mais fraco do cluster que possui alguma fronteira com um país inimigo.

4.4 ANGRY

O Angry é um agente com característica agressiva. Ele tenta gastar todos as suas tropas para ganhar rápido o jogo. Os seus métodos podem ser descritos da seguinte forma:

- Mobilização: o seu método de mobilização coloca as tropas novas no país aliado com maior número de tropas inimigas somadas em suas fronteiras.

- Ataque: o seu método de ataque seleciona para cada país aliado o país vizinho inimigo mais fraco e o ataca. Quando um ataque é iniciado ele é realizado até que o país inimigo seja conquistado ou que reste apenas uma tropa no país aliado atacante.

- Conquista: o método checa qual dos dois países tem mais tropas inimigas ao redor e deixa todas as tropas com aquele país.

- Fortificação: neste método, o agente faz uma lista com países aliados com mais de uma tropa. Então ele percorre essa lista procurando por todos os países aliados conectados com os países da lista. Eles podem tanto ser vizinhos quanto estarem conectados através de outros países aliados. No momento que ele encontra um país aliado com mais de uma tropa conectado a um país com um número maior de inimigos em volta, ele realiza a fortificação, movendo todos as tropas para o país em perigo.

4.5 MONTE CARLO

O agente com BAMC herda suas funções de mobilização, conquista e fortificação do agente Cluster. A única função própria do agente é a função de ataque, onde a BAMC é aplicada para que se encontre ao longo dos jogos as melhores ações a serem tomadas dependendo do estado. Uma ideia similar foi proposta por J. Lozano (LOZANO; BRATZ, 2012).

Sempre que é a vez do agente atacar, ele gera um nodo $G = \{S, L, v, n\}$. S é o estado atual do jogo, representado por uma matriz $i \times j$, sendo i o número de jogadores e j o número de países existente no jogo. Na implementação do jogo utilizada nesta monografia, o número de jogadores é sempre dois, logo o número de linhas da matriz será sempre dois. A matriz armazena o número de tropas que o jogador i tem no país j. L é uma lista com todos os nodos filhos de G acompanhados pela ação que G deve tomar para chegar neles. v representa o valor do nodo, que é calculado pela diferença de vitórias e derrotas que aquele nodo já participou (pode assumir um valor negativo). Por último, n é o número de vezes que aquele nodo já foi visitado.

Apenas um ataque não tem um impacto tão grande no jogo, já que é possível atacar um país apenas uma vez e então desistir do ataque, não realizando ganhos ou perdas significativas. Isso torna mais difícil o aprendizado do agente, já que é mais complexo avaliar se o resultado final do jogo foi devido a decisão tomada ou algum outro elemento. Por esse motivo, como sugerido por J. Lozano (LOZANO; BRATZ, 2012), decidiu-se limitar a ação de atacar do agente Monte Carlo. Sendo assim, quando um ataque é ordenado, ele será realizado até o final, ou seja, até o país alvo ser conquistado ou até o país aliado ficar com apenas uma tropa. Desse modo, as ações tomadas terão impactos maiores, resultando em conquistas ou perdas mais significativas.

O estado inicial do agente se dá a partir do primeiro turno de ataque. Nele verifica-se se o nodo atual é um nodo conhecido ou não. Caso não seja um nodo conhecido, ele será adicionado à árvore do jogo e será iniciada a etapa de simulação do restante do jogo. Já caso seja um nodo conhecido, ele irá percorrer os nodos filhos, escolhendo aquele com maior valor de acordo com a fórmula UCT, até chegar em uma folha (nodo sem filhos), tendo 5% de chance de começar uma *exploration* ao longo deste caminho. Essa chance de realizar *exploration* sem estar em um nodo folha não é original da BAMC, ela foi adicionada para que haja uma variabilidade maior na exploração de ações no jogo. Chegando em um nó folha, o agente irá gerar um novo nodo nunca antes explorado. A geração desse nodo é feita realizando uma ação possível aleatória, seja de ataque, escolhendo qualquer país apto a atacar para iniciar um ataque contra um vizinho inimigo aleatório, ou de passar o turno. A partir desse novo nodo será feita a simulação do resto do jogo, onde não são mais armazenados os nodos gerados. Foi escolhida uma abordagem de *heavy rollout* para a simulação do jogo, onde as escolhas de ataque tomadas nessa etapa são herdadas da

função de ataque do agente Cluster. Com o jogo finalizado, ocorrerá o *backpropagation*, que adiciona um valor ($+1$ em caso de vitória e -1 em caso de derrota) ao v de cada nodo percorrido antes da etapa de simulação, incrementando também o contador n de cada nodo.

Nota-se que, a cada partida alguns outros nodos já existentes são percorridos e avaliados, enquanto que apenas um nodo novo é adicionado à arvore. Isso faz com que o aprendizado seja lento, porém mais preciso.

Para o treinamento do agente com BAMC utilizou-se de um arquivo json para armazenar todos os nodos e conexões da árvore. A cada partida o agente pega as informações da árvore, joga a partida e atualiza o arquivo no final com as novas informações descobertas. Pode-se ver a estrutura do arquivo json utilizado para armazenamento da árvore no pseudocódigo no apêndice C.

5 EXPERIMENTOS E RESULTADOS

Cada agente passou por 50 partidas teste[1] contra todos os outros agentes, tanto como *player 1* quanto como *player 2*. A análise do ponto de vista do *player 1* e do *player 2* é importante, pois certas estratégias podem variar o percentual de vitórias dependendo de o agente ter sido o primeiro a jogar ou o último. Foram analisados a porcentagem de vitória, número de tropas finais do vencedor, média de escritas nos arquivos json que mantém os estados do jogo e dos jogadores e tempo médio de execução.

O treinamento do agente com BAMC foi realizado contra o agente Cluster, tanto como *player 1* quanto como *player 2*. Um total de 2000 partidas foram realizadas.

Como o Risk é um jogo que contém aleatoriedade, ele não tem um estado inicial fixo, podendo variar os países possuídos por cada jogador, assim como o número de tropas neles. Como a etapa de *exploitation* depende de começar o jogo em um estado já conhecido, nos testes iniciais é normal que haja apenas *exploration* para então, após algumas centenas de execuções, encontrar algum estado inicial repetido e começar aos poucos a fazer etapas de *exploration*. Dessa forma, estima-se que ocorrerá uma *exploitation* significativa da árvore do jogo após um número extremamente grande de execuções.

Por esse motivo, para que seja possível demonstrar o funcionamento da etapa de *exploitation* de forma mais clara, além de treinar o agente da forma convencional, optou-se por treiná-lo em partidas com o estado inicial fixo. Dessa forma, desde a segunda partida do jogo já há *exploitation* e, consequentemente, um aprendizado mais rápido por conta da limitação no espaço de estados iniciais do jogo. Lembrando que, apesar do espaço de estados iniciais estar limitado somente a um estado, o espaço total de estados continua sendo infinito, já que ainda não há limitação de tropas em campo assim como de ações a serem realizadas. Dessa maneira, o algoritmo foca em buscas na sub-árvore gerada a partir do espaço inicial fixado. O estado inicial fixo está disponível no apêndice D. Essa técnica de treino, por sacrificar parte da capacidade de generalização do agente, é utilizada apenas com o objetivo de ilustrar de forma mais clara o aprendizado do mesmo. Foram realizadas 4000 partidas de treino como *player 1* e o mesmo número como *player 2*.

5.1 MÉDIAS DE VITÓRIAS

Nas tabelas 5.1 e 5.2 estão as porcentagens de vitórias dos agentes nomeados nas linhas em relação aos agentes das colunas. Na tabela 5.1 os agentes das linhas são os

[1] Os testes foram realizados em um processador Intel(R) Core(TM) i7-6500U CPU dual core com núcleos de 2.50GHz e 2.60 GHz, sistema operacional Windows Home de 64 bits e 8 GB de memória RAM DDR3. Os resultados completos podem ser encontrados no repositório *Risk-Agents* (Gargano Ferrari, René, 2022).

primeiros a jogar, já na tabela 5.2 eles são os últimos a jogar.

Tabela 5.1 – Porcentagens de Vitórias como Player 1

	Random	**Angry**	**Cluster**	**Monte Carlo**
Random	15%	19%	25%	40%
Angry	98%	84%	94%	92%
Cluster	96%	58%	64%	73%
Monte Carlo	96%	41%	72%	76%

Fonte: Autor

Tabela 5.2 – Porcentagens de Vitórias como Player 2

	Random	**Angry**	**Cluster**	**Monte Carlo**
Random	85%	2%	4%	4%
Angry	81%	16%	42%	59%
Cluster	75%	6%	36%	28%
Monte Carlo	60%	8%	27%	24%

Fonte: Autor

Através da análise das tabelas é possível notar que, na implementação do jogo utilizada nesta monografia, há uma clara vantagem em se jogar como o *player 1*. O único agente que obteve vantagem contra ele mesmo como *player 2* foi o agente Random. É visível que, tirando o Random e a situação de Monte Carlo contra Angry, todos os outros agentes têm uma chance de vitória acima de 50% quando jogam primeiro. Essa vantagem é mais clara quando analisadas as tabelas 5.3 e 5.4, que mostram a porcentagem média de vitórias de cada agente respectivamente como *player 1* e *player 2*.

Além disso, o agente Monte Carlo como *player 1* tem um aumento de vitórias em relação ao seu agente pai, Cluster, no caso de Monte Carlo versus Cluster e Monte Carlo versus Monte Carlo. Já em Monte Carlo versus Angry ele acaba tendo um desempenho pior. Como *player 2* ele acaba tendo um desempenho pior que seu pai, exceto em Monte Carlo versus Angry, onde ele tem 2% a mais de vitórias, um resultado quase igual.

Tabela 5.3 – Médias de Vitórias como Player 1

	Média de Vitórias
Random	24.5%
Angry	92%
Cluster	72%
Monte Carlo	71.25%

Fonte: Autor

Tabela 5.4 – Médias de Vitórias como Player 2

	Média de Vitórias
Random	23.5%
Angry	49.5%
Cluster	36.25%
Monte Carlo	29.75%

Fonte: Autor

5.2 QUANTIDADE DE TROPAS AO VENCER

Nas tabelas 5.5 e 5.6 é ilustrado o número de tropas médias pertencentes ao vencedor ao final das partidas, sendo os vencedores representados pelos agentes das linhas. Esse dado não está diretamente relacionado ao desempenho do agente, mas abre espaço para debate sobre as diferentes estratégias adotadas pelos mesmos.

Tabela 5.5 – Média de Tropas Finais ao Vencer como Player 1

	Random	**Angry**	**Cluster**	**Monte Carlo**
Random	339.5	297.2	380.84	456.65
Angry	95.61	112.28	102.74	97.21
Cluster	98.53	86.25	95.06	96.41
Monte Carlo	92.08	88.85	94.04	95.07

Fonte: Autor

Tabela 5.6 – Média de Tropas Finais ao Vencer como Player 2

	Random	**Angry**	**Cluster**	**Monte Carlo**
Random	89.11	386	425.5	674
Angry	91.43	109.75	99.73	101.1
Cluster	97.28	88	93.38	94.39
Monte Carlo	99.36	104.5	91.62	89.58

Fonte: Autor

Nota-se que o agente Random possui um número médio de tropas maior. Isso acontece por o agente não utilizar de uma estratégia definida, o que o faz demorar mais para vencer. Já os outros agentes compartilham médias parecidas de tropas. O agente Cluster possui uma média menor que a do agente Angry, mesmo teoricamente tendo uma estratégia mais conservadora. Isso pode indicar que talvez a estratégia de ataque do Cluster seja agressiva demais e ainda tenha espaço para ser aprimorada para melhor cumprir a sua proposta de proteger e expandir grandes blocos de países aliados pelo mapa. Essas possíveis mudanças afetariam também o agente Monte Carlo, filho do Cluster.

5.3 TEMPO DE EXECUÇÃO MÉDIO

Na tabela 5.7 é mostrado o tempo médio de execução das partidas. Isso é importante pois o jogo ocorre através de trocas de arquivos json, o que diminui consideravelmente a velocidade de execução do jogo devido ao alto número de *inputs* e *outputs*. O treinamento do agente Monte Carlo acaba dependendo bastante do tempo de execução de cada partida, pois precisa de muitas partidas para explorar diferentes nodos da árvore do jogo.

Tabela 5.7 – Média de tempo de execução (s)

	Random	**Angry**	**Cluster**	**Monte Carlo**
Random	6.32	7.08	11.19	12.10
Angry	4.6	7.01	4.6	4.63
Cluster	6.75	7.13	8.42	8.49
Monte Carlo	7.66	6.0	8.09	6.67

Fonte: Autor

Nota-se que alguns dos maiores tempos estão atrelados a presença do agente Random devido a sua aleatoriedade. Já o Angry detém uma média de tempo menor por

ser mais agressivo e tentar ganhar o jogo logo no início gastando todos os as suas tropas disponíveis.

O tempo médio do agente Monte Carlo como *player 1* foi de 7.1s e como *player 2* foi de 7.97s. Com essas médias de tempo é possível realizar aproximadamente 470 partidas por hora em um ambiente ideal. No entanto, durante os testes realizados nesta monografia, por conta da alta frequência de troca de arquivos, foi adicionado um tempo de espera de alguns segundos entre as execuções para se ter certeza de que os processos foram totalmente finalizados antes do inicio de uma nova partida, evitando assim o corrompimento dos arquivos json e, consequentemente, tornando mais lento o processo de aprendizado.

5.4 TROCA DE ARQUIVOS MÉDIA

A tabela 5.8 mostra a média de escrita dos arquivos com os dados do jogo e dos jogadores. Esses arquivos são reescritos após toda ação dos agentes.

Tabela 5.8 – Média de troca de arquivos

	Random	**Angry**	**Cluster**	**Monte Carlo**
Random	313.11	358.50	581.25	667.04
Angry	223.18	256.58	213.52	230.31
Cluster	368.52	320.03	432.62	408.98
Monte Carlo	430.92	310.88	405.04	380.96

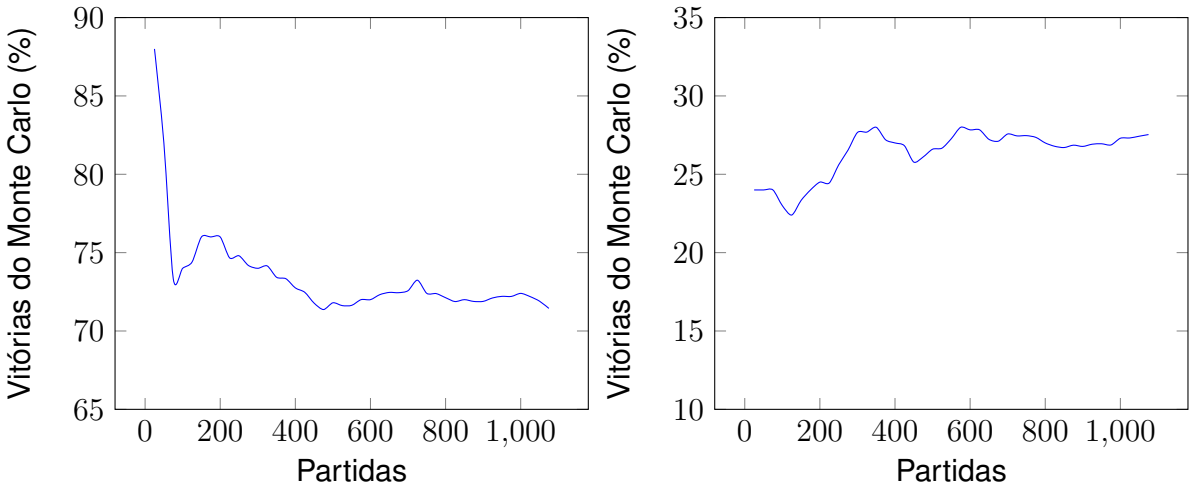
Fonte: Autor

É possível identificar que há um grande número de *inputs* e *outputs* ao longo das partidas. Esse número alto de trocas é o que compromente a eficiência do jogo e dos agentes. O Random novamente tem uma média maior devido a demorar mais tempo para vencer. Ainda há espaço para otimizações no sistema de troca de informações jogo-agente.

5.5 TREINAMENTO DO AGENTE MONTE CARLO

A figura 5.1 mostra através de dois gráficos a porcentagem de vitórias do agente Monte Carlo ao longo de aproximadamente 2000 partidas contra o agente Cluster. As primeiras 1000 partidas, representadas pelo gráfico da esquerda, foram com o Monte Carlo jogando como *player 1* e as próximas 1000, representadas pelo gráfico da direita, foram com o Monte Carlo como *player 2*, guardando os dois aprendizados na mesma árvore.

Figura 5.1 – Esquerda: Monte Carlo vs. Cluster. Direita: Cluster vs. Monte Carlo



Fonte: Autor

De início, até aproximadamente a milésima partida, vê-se que o agente tem uma queda na porcentagem de vitórias. Isso acontece porque durante a etapa de simulação, devido ao *heavy rollout*, a estratégia de ataque herdada do Cluster é utilizada. No início do treino, praticamente só é feita simulação para a validação de nós iniciais. Ao longo das partidas, as sub-árvores geradas vão aumentando e a etapa de *exploitation* aumenta junto, enquanto que a de simulação diminui. Sendo assim a estratégia herdada do Cluster vai sendo substituída aos poucos pela estratégia aprendida pelo Monte Carlo, que aparentemente possui uma chance de vitória menor.

Já a partir do momento que o Monte Carlo começa a jogar como *player 2*, apesar do agente iniciar com uma porcentagem de vitória extremamente baixa devido a desvantagem, há um aumento na porcentagem de vitórias ao longo das partidas jogadas. Isso indica que há evolução por parte do agente.

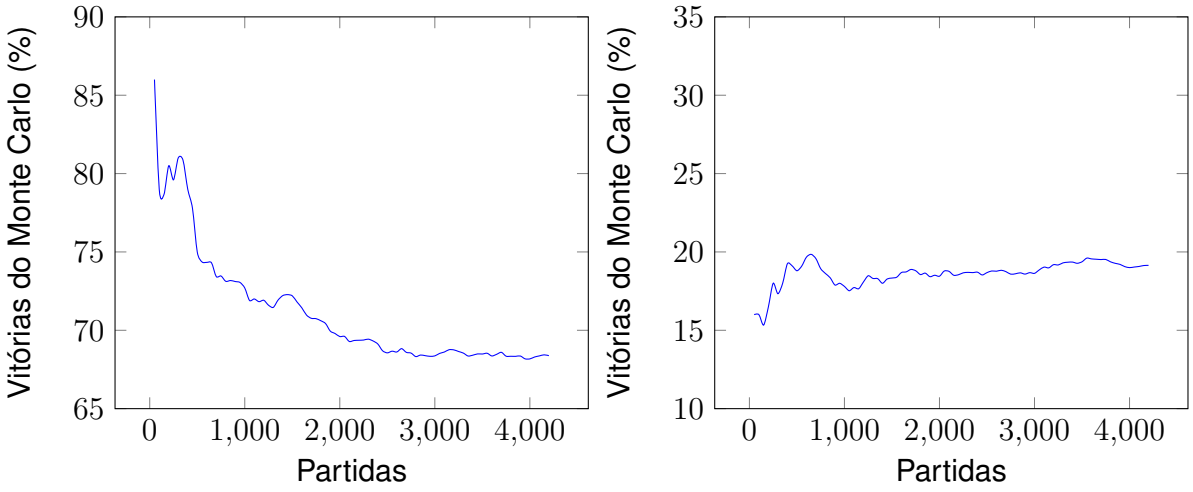
5.6 TREINAMENTO DO AGENTE MONTE CARLO COM INÍCIO FIXADO

Como já dito anteriormente, o jogo Risk possui um espaço de estados extremamente grande, já que os estados dependem do número de tropas em campo, o qual não possui uma limitação. Por esse motivo, torna-se difícil a visualização do aprendizado do agente Monte Carlo em um curto período de tempo de treino, já que a chance de cair em um estado inicial conhecido é pequena nas primeiras partidas e ele depende dela para iniciar a etapa de *exploitation*.

Pensando nisso, resolveu-se realizar testes com o estado inicial do jogo fixado, fazendo com que a etapa de *exploitation* ocorra desde o início do treino. Os dois treinos

foram realizados separadamente, cada um com a sua própria árvore sendo gerada do zero. Ambos os treinos duraram aproximadamente 4000 partidas. Na figura 5.2 é possível visualizar os resultados.

Figura 5.2 – Início Fixo. Esquerda: Monte Carlo vs. Cluster. Direita: Cluster vs. Monte Carlo



Fonte: Autor

Nota-se que, como no primeiro treinamento, realizado com estados iniciais aleatórios, o aprendizado resulta em uma evolução na taxa de vitória quando o agente está jogando como *player 2*. Já como *player 1*, a taxa acaba caindo conforme o aprendizado do agente. Isso nos mostra que o agente é bem sucedido ao aprender a compensar a desvantagem a ele imposta por estar jogando como *player 2*.

6 CONCLUSÃO E TRABALHOS FUTUROS

Esta monografia teve como objetivo o desenvolvimento, análise e comparação de diferentes agentes para o jogo de tabuleiro americano Risk, sendo um com comportamento aleatório, dois com heurísticas baseadas nos agentes do jogo Lux Delux (SILLYSOFT, 2002) e, finalmente, um agente de RL treinado com a técnica de BAMC.

A implementação dos agentes foi feita voltada para uma implementação em Python do jogo Risk (PAVIN, 2022), que possibilita a criação de agentes em qualquer linguagem capaz de realizar modificações em arquivos json. O uso desta implementação, no entanto, por conter grande número de *inputs* e *outpus* acaba sacrificando desempenho na execução do jogo e dos agentes, o que fez com que o treinamento e os testes dos agentes fosse um pouco mais demorado e sofresse com eventuais problemas de falhas nos arquivos json de comunicação.

Estas falhas tornaram um pouco mais complicado o desenvolvimento dos agentes e treinamento do agente Monte Carlo. Uma possível solução para isso é a migração do jogo base para a linguagem C++, onde talvez seja possível ter um controle mais fino sobre essas etapas, além de um aumento na performance do jogo. Junto a isso, aprimoramentos em funções de busca utilizadas tanto pelos agentes quanto pelo jogo podem ser implementados. A solução também pode envolver o desenvolvimento de outros protocolos para a comunicação jogo-agente que ainda possibilitem a produção de agentes em qualquer linguagem com poder de modificação de arquivos json.

Os testes realizados nesta monografia mostraram que há uma vantagem clara em se jogar como *player 1* na implementação do jogo utilizada. Praticamente todos os agentes, com excessão do Random, obtiveram taxas médias de vitória maiores que 50% quando ocupavam o lugar do primeiro jogador.

Além disso, sobre o treinamento do agente Monte Carlo, percebeu-se aprendizado ao longo das partidas de treino. Como *player 1*, apesar dele começar com vantagem, há uma diminuição na taxa de vitória ao longo das iterações. Já como *player 2*, apesar da enorme desvantagem imposta pela posição de segundo jogador, o agente demonstra melhora na sua taxa de vitória ao longo das várias partidas de treino. Mesmo o aprendizado tendo sido notado, a evolução na taxa de vitória ainda é pequena quando acontece, mostrando assim que ainda há espaço para melhorias na metodologia utilizada para o aprendizado do agente.

Uma forma de otimizar o aprendizado do agente Monte Carlo é, futuramente, procurar generalizar mais os estados do jogo, pois, atualmente, apenas uma tropa de diferença já é o suficiente para se criar um estado completamente novo. Fazer isso iria juntar em um estado só vários estados extremamente parecidos que teriam a mesma solução, ou soluções muito parecidas, diminuindo assim o espaço de estados do jogo e aumentando

a performance do agente. Na implementação atual, a variabilidade de nós filhos na árvore é resultado da taxa de 5% de chance de se iniciar uma *exploration* em um nó que não é folha. Para que se tenha uma variabilidade mínima garantida de ramificações na árvore, aconselha-se que seja implementado uma constante de ramificação, que fará com que toda folha gere no mínimo um número fixo de filhos.

Além disso, nesta monografia, as regras foram simplificadas para facilitar a implementação tanto do jogo quanto dos agentes. Futuramente, para uma implementação mais fiel ao jogo original, deve-se acrescentar a possibilidade de escolha de países por parte dos agentes na etapa de *setup*, além da implementação do sistema de cartas presente no jogo físico.

Sobre a implementação dos agentes, é possível melhorar o desempenho do Cluster trabalhando em seu método de ataque, que de acordo com os testes, ainda se mostra muito agressivo considerando a proposta do agente. Ademais, há espaço para a implementação de mais agentes baseados em heurísticas do jogo Lux Delux (SILLYSOFT, 2002), sendo eles um agente baseado na heurística do Shaft, herdando funções do Cluster e, caso implementada a mecânica de cartas, a implementação de um agente baseado na heurística do Quo, herdeiro do Shaft, considerado um dos mais poderosos pelos desenvolvedores do jogo.

REFERÊNCIAS BIBLIOGRÁFICAS

BERNER, C. et al. Dota 2 with large scale deep reinforcement learning. **arXiv preprint arXiv:1912.06680**, 2019.

BROWNE, C. B. et al. A survey of monte carlo tree search methods. **IEEE Transactions on Computational Intelligence and AI in games**, IEEE, v. 4, n. 1, p. 1–43, 2012.

CARR, J. Using graph convolutional networks and td (λ) to play the game of risk. **arXiv preprint arXiv:2009.06355**, 2020.

COSTALBA, M.; KIISKI, J.; ROMSTAD, T. **Stockfish**. 2008. <https://stockfishchess.org/>.

DUAN, Y.; EDWARDS, J. S.; DWIVEDI, Y. K. Artificial intelligence for decision making in the era of big data–evolution, challenges and research agenda. **International Journal of Information Management**, Elsevier, v. 48, p. 63–71, 2019.

Gargano Ferrari, René. **Risk-Agents**. 2022. Acesso em 01 fev. 2022. Disponível em: <https://github.com/rgferrari/Risk-Agents>.

GEORGIOU, H. Risk board game-battle outcome analysis. **An Example of game-theoretic approaches to analyze simple board games and evaluate globally optimal strategies,(15. 12. 2016)**, 2004.

GIBSON, R.; DESAI, N.; ZHAO, R. An automated technique for drafting territories in the board game risk. In: . [S.l.: s.n.], 2010.

HASBRO. **Risk**. 1959. <https://www.hasbro.com/common/instruct/risk.pdf>.

JARDIM, J. P. C. R. et al. An implementation of the 7 wonders board game for ai-based players. 2020.

LOZANO, J.; BRATZ, D. A risky proposal : Designing a risk game playing agent. In: . [S.l.: s.n.], 2012.

OLSSON, F. **A multi-agent system for playing the board game risk**. 2005. Dissertação (Mestrado), 2005.

PAVIN, T. **Uma implementação de Risk para competição e aprendizado em IA**. 2022. Monografia (Trabalho de Graduação), 2022.

Pavin, Thiago and Gargano Ferrari, René. **Risk_Implementation**. 2022. Acesso em 01 fev. 2022. Disponível em: <https://github.com/ThiagoPavin/Risk-Implementation>.

RUSSELL, S. J.; NORVIG, P. **Artificial Intelligence: a modern approach**. 3. ed. [S.l.]: Pearson, 2009.

SILLYSOFT. **Lux Delux**. 2002. <https://sillysoft.net/lux/>.

SILVER, D. et al. Mastering the game of go with deep neural networks and tree search. **nature**, Nature Publishing Group, v. 529, n. 7587, p. 484–489, 2016.

VINYALS, O. et al. **AlphaStar: Mastering the Real-Time Strategy Game StarCraft II**. 2019. <https://deepmind.com/blog/alphastar-mastering-real-time-strategy-game-starcraft-ii/>.

WOLF, M. An intelligent artificial player for the game of risk. **Unpublished doctoral dissertation. TU Darmstadt, Knowledge Engineering Group, Darmstadt Germany. http://www. ke. tu-darmstadt. de/bibtex/topics/single/33**, 2005.

APÊNDICE A – DADOS DO JOGO E DOS JOGADORES

```
country_object = {
    (str)<nome do país>: {
        "neighbours":
            [<strings com nomes dos países vizinhos>],
        "owner":
            (int)<identificador do dono do país>,
        "n_troops":
            (int)<número de tropas no país>
    }
}
countrys_borders = {
    (str)<nome do país aliado>:
        [<strings com nomes dos países vizinhos que são
        inimigos>]
}
connection = {
    (str)<nome do país aliado>: true ou false
}
countrys_connections = {
    (str)<nome do país aliado>: {
        connection,           // Declarado acima
        connection,
        ...
    }
}
continent_object = {
    (str)<nome do continente>: {
        "owner":
            (int)<identificador do dono do país>,
        "extra_armies":
            (int)<número de tropas bônus fornecidas pelo continente>,
        "countries":
            [<strings com nomes dos países pertencentes ao
            país>]
    }
}
```

```
pn_state.json = {
    "count":
        (int)<número de escritas no arquivo>,
    "id":
        (int)<id do jogador>,
    "n_new_troops":
        (int)<número de tropas disponíveis para serem alocadas no
        tabuleiro>,
    "n_total_troops":
        (int)<número total de tropas do jogador dispostas no tabuleiro>,
    "enemy_n_total_troops":
        (int)<número total de tropas do inimigo dispostas no tabuleiro>,
    "state":
        "mobilizing" ou "attacking" ou "conquering" ou "fortifying" ou
        "waiting",
    "countries_owned":
        [<strings com nomes dos países pertencentes ao
        jogador>],
    "countries_data": {
        country_object,       // Declarado acima
        country_object,
        ...
    },
    "border_countries":{
        countrys_border,      // Declarado acima
        countrys_border,
        ...
    },
    "connection_matrix":{
        countrys_connections,  // Declarado acima
        countrys_connections,
        ...
    },
    "continents_data":{
        continent_object,     // Declarado acima
        continent_object,
        ...
    }
}
```

APÊNDICE B – CHAMADA DE AÇÃO DO AGENTE

```
command_1 = {
    "name": "attack",
    "args": [(int)<número de dados>, (str)<país atacante>,
    (str)<país alvo>]
}
command_2 = {
    "name": "move_troops",
    "args": [(int)<número de tropas>, (str)<país remetente>,
    (str)<país destino>]
}
command_3 = {
    "name": "set_new_troops",
    "args": [(int)<número de tropas>, (str)<país destino>]
}
command_4 = {
    "name": "pass_turn",
    "args": []
}

pn_call.json =
{
    "id":
        <id do jogador>,
    "count":
        <número de escritas no arquivo>,
    "command":
        // Declarados acima
        command_1 ou command_2 ou command_3 ou command_4
}
```

APÊNDICE C – ESTRUTURA DA ÁRVORE DO AGENTE MONTE CARLO

```
action_1 = "pass"
action_2 = (str)<país atacante>, (str)<país atacado>

montecarlo_tree.json = {
    (str)<id do nodo>: {
        "state": {
            "player": {
                (str)<país aliado>: (int)<número de tropas>,
                (str)<país aliado>: (int)<número de tropas>,
                ...
            },
            "enemy": {
                (str)<país inimigo>: (int)<número de tropas>,
                (str)<país inimigo>: (int)<número de tropas>,
                ...
            }
        },
        "leafs": [
            // action_1 e action_2 declarados acima
            [(str)<id do filho>, action_1 ou action_2],
            [(str)<id do filho>, action_1 ou action_2],
            ...
        ],
        "n_visits": (int)<número de vezes que o nodo foi visitado>
        "value": (int)<saldo entre vitórias e derrotas com o nodo>
    }
    ...
}
```

APÊNDICE D – ESTADO INICIAL FIXO UTILIZADO NO TESTE

```
fixed_state = {
    "Alaska": {"owner_id": 1, "n_troops": 2},
    "Alberta": {"owner_id": 2, "n_troops": 2},
    "Ontario": {"owner_id": 1, "n_troops": 2},
    "Western America": {"owner_id": 2, "n_troops": 2},
    "Eastern America": {"owner_id": 1, "n_troops": 2},
    "Quebec": {"owner_id": 2, "n_troops": 2},
    "Central America": {"owner_id": 1, "n_troops": 2},
    "Greenland": {"owner_id": 2, "n_troops": 2},
    "Northwest America": {"owner_id": 1, "n_troops": 2},
    "Brazil": {"owner_id": 2, "n_troops": 2},
    "Venezuela": {"owner_id": 1, "n_troops": 2},
    "Peru": {"owner_id": 2, "n_troops": 2},
    "Argentina": {"owner_id": 1, "n_troops": 2},
    "Western Australia": {"owner_id": 2, "n_troops": 2},
    "Eastern Australia": {"owner_id": 1, "n_troops": 2},
    "Indoneasia": {"owner_id": 2, "n_troops": 2},
    "Papua New Guinea": {"owner_id": 1, "n_troops": 2},
    "Ukraine": {"owner_id": 2, "n_troops": 2},
    "Skandinavia": {"owner_id": 1, "n_troops": 2},
    "Iceland": {"owner_id": 2, "n_troops": 2},
    "Great Britain": {"owner_id": 1, "n_troops": 2},
    "Northern Europe": {"owner_id": 2, "n_troops": 2},
    "Western Europe": {"owner_id": 1, "n_troops": 2},
    "Southern Europe": {"owner_id": 2, "n_troops": 2},
    "Yakutsk": {"owner_id": 1, "n_troops": 2},
    "Siberia": {"owner_id": 2, "n_troops": 2},
    "Kamchatka": {"owner_id": 1, "n_troops": 2},
    "Irkutsk": {"owner_id": 2, "n_troops": 2},
    "Ural": {"owner_id": 1, "n_troops": 2},
    "Japan": {"owner_id": 2, "n_troops": 2},
    "Mongolia": {"owner_id": 1, "n_troops": 2},
    "China": {"owner_id": 2, "n_troops": 2},
    "Middle East": {"owner_id": 1, "n_troops": 2},
    "India": {"owner_id": 2, "n_troops": 2},
    "Siam": {"owner_id": 1, "n_troops": 2},
```

```
    "Afganistan": {"owner_id": 2, "n_troops": 2},
    "Congo": {"owner_id": 1, "n_troops": 2},
    "East Africa": {"owner_id": 2, "n_troops": 2},
    "Egypt": {"owner_id": 1, "n_troops": 1},
    "Madagascar": {"owner_id": 2, "n_troops": 1},
    "North Africa": {"owner_id": 1, "n_troops": 1},
    "South Africa": {"owner_id": 2, "n_troops": 1}
}
```