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

Thiago Bellotti Pavin

UMA IMPLEMENTAÇÃO DE RISK PARA COMPETIÇÃO E APRENDIZADO EM IA

Santa Maria, RS
2022

Thiago Bellotti Pavin

UMA IMPLEMENTAÇÃO DE RISK PARA COMPETIÇÃO E APRENDIZADO EM IA

Trabalho de Conclusão de Curso apresentado ao Curso de Graduação em Ciência da Computação da Universidade Federal de Santa Maria (UFSM, RS), como requisito parcial para obtenção do grau de **Bacharel em Ciência da Computação**. Defesa realizada por videoconferência.

ORIENTADOR: Prof. Joaquim Vinicius Carvalho Assunção

Santa Maria, RS
2022

Thiago Bellotti Pavin

UMA IMPLEMENTAÇÃO DE RISK PARA COMPETIÇÃO E APRENDIZADO EM IA

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

Aprovado em 16 de fevereiro de 2022:

_____

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

_____

Cesar Tadeu Pozzer, Dr. (UFSM) (videoconferência)

_____

Giliane Bernardi, Dr. (UFSM) (videoconferência)

Santa Maria, RS
2022

RESUMO

UMA IMPLEMENTAÇÃO DE RISK PARA COMPETIÇÃO E APRENDIZADO EM IA

AUTOR: Thiago Bellotti Pavin
ORIENTADOR: Joaquim Vinicius Carvalho Assunção

Junto com a evolução da inteligencia artificial (IA), os jogos tem se mostrado cada vez mais um importante ambiente para treino e testes nesta área. Isso se dá por serem ambientes controlados e de fácil repetição. Além disso, eles também têm sido amplamente utilizados como ferramenta de educação na área de IA. Para complementar esse ensino, competições também são utilizadas. Pois além de gerar interesse e motivar, conseguem produzir conteúdos para os próximos estudantes que forem estudar ou pesquisar sobre a área. Pensando nisso, este trabalho propõe a implementação do jogo de tabuleiro Risk, com uma arquitetura simplificada e direcionada a competição entre agentes (IA), onde a comunicação entre o jogo e o agente será feita através de arquivos *JSON*, permitindo a implementação dos agentes em diferentes linguagens com suporte a leitura e escrita desse tipo de arquivo. Risk é um jogo totalmente observável, estocástico e baseado em turno, essas características permitem a criação de diversos métodos de IA, dos simples até os mais complexos. O objetivo é desenvolver um ambiente com interface visual para competições de IA, que seja simples e de fácil implementação dos seus agentes, onde estudantes com diversos níveis de conhecimento possam utilizar com uma grande liberdade.

Palavras-chave: Risk, Jogos de tabuleiro, Competição, Aprendizagem, Inteligência Artificial, Agentes.

ABSTRACT

AN IMPLEMENTATION OF RISK FOR COMPETITION AND LEARNING IN AI

AUTHOR: Thiago Bellotti Pavin
ADVISOR: Joaquim Vinicius Carvalho Assunção

Along with the evolution of Artificial Intelligence (AI), the games have increasingly proven an important training and testing environment in this area. This is because they are controlled environments and easy to repeat. In addition, they have been widely used as an AI education tool. To complement this teaching, competitions are also used, because in addition to generate interest and motivation, they are able to produce content for the next students who are going to study or research the area of AI. With this in mind, this work propose an implementation of the board game Risk, with a simplified architecture and aimed at competitions between agents (AI), where the communication between game and agents will be done through *JSON* files, allowing agents implementation in different languages with support for reading and writing on this type of file. Risk is a fully observable, stochastic and turn-based game, these characteristics allow the creation of different AI methods, from the simple to the most complex. The goal is to develop an environment with a visual interface for AI competitions, which is simple and easy to implement its agents, where students with different levels of knowledge can use it with great freedom.

Keywords: Risk, Board Games, Competition, Learning, Artificial Intelligence, Agents.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

IA	Inteligência Artificial
AI	Artificial Intelligence
$JSON$	JavaScript Object Notation
IBM	International Business Machines Corporation

SUMÁRIO

1 INTRODUÇÃO

A Inteligência Artificial (IA) é uma área da computação que existe há décadas. Alan Turing em 1950 publicou o artigo *Computing Machinery and Intelligence* (TURING, 1950) na revista *Mind*, onde introduziu o conceito do *Turing Test*, que é um teste para identificar se um máquina é capaz de pensar ou se comportar como um humano.

Atualmente a IA tem crescido e ganhado bastante importância. Isso pode ser atribuído pelo aumento do poder computacional disponível, que possibilitou a sua aplicação nas mais diversas áreas, como compreensão de imagens e fala, detecção de emoção, carros autônomos, pesquisa na web e jogos.

Existem diversas técnicas para a implementação de uma IA, que vão de técnicas relativamente simples como a árvore de decisão até às bastante complexas, como as redes neurais profundas, mas mesmo a mais simples pode não ser tão fácil e intuitiva de aprender e aplicar, o que acaba desmotivando as pessoas de continuar a estudar sobre a área.

Pensando nesse problema, os videogames juntamente com competições estão sendo utilizados como formas de ensino na computação (CARPIO et al., 2014a; ŚWIECHOWSKI, 2020; SALTA; PRADA; MELO, 2020). As competições geram motivação e nos fazem querer ir atrás de soluções melhores, que no final podem gerar bons conteúdos e de fácil acesso para novos estudantes. Já os videogames capturam o interesse desses estudantes, que na maioria das vezes já têm contato com estes antes do início de sua educação em programação. Os jogos de computador costumam ser um dos motivos que os atraem para a área da computação.

Para a área de IA, os jogos têm uma grande importância, pois desde o nascimento da ideia de inteligência artificial, eles têm ajudado no seu progresso de pesquisa (TOGELIUS, 2018). Os jogos se tornaram um dos mais importantes ambientes de teste para a área de IA, as principais razões são que eles são relativamente baratos, facilmente repetíveis e representam ambientes controlados.

Além disso, eles podem imitar ambientes do mundo real, podendo refletir os seus problemas e assim aprender e analisar estes ambientes sem gerar custos e riscos com testes em ambientes reais. Isso gera, por exemplo, avanços na saúde, melhora na inteligência de carros autônomos e progresso em cidades inteligentes e sustentáveis.

Os primeiros jogos que foram conhecidos globalmente que são utilizados para testes em Inteligência Artificial (IA) foi damas e xadrez. Em 1996, o *Deep Blue* (NEWBORN, 1996) da IBM derrotou o campeão mundial no xadrez e em 2016, o algoritmo *AlphaGo* (SILVER et al., 2016) do Google venceu um dos melhores jogadores do jogo de tabuleiro chinês Go.

Damas, Xadrez e Go têm algumas características em comum, eles não têm fatores aleatórios, cada jogador tem número fixo de jogadas por turno e cada ação causa uma

consequência específica, é possível ter uma visão completa de suas peças e posições e são simples para humanos aprender suas regras, mas são complexos para se tornar um expert. Estas características facilitam a implementação de IAs, e por consequência já existem bastantes pesquisas nos dias atuais sobre IA em jogos com essas características. Já, jogos como Risk, são estocásticos e seus jogadores podem fazer diversas jogadas em um único turno, características que não são muito bem exploradas atualmente.

Pensando nestas duas abordagens de ensino (videogames e competições) e nas características diferentes do jogo Risk, este trabalho propõe a implementação do jogo de tabuleiro Risk (HASBRO, 1993), com a arquitetura simplificada e direcionada a utilização de IA (agentes).

Para IA, agentes definem qualquer coisa que pode ser vista operando de forma autônoma, percebendo um ambiente por meio de sensores e atuando no mesmo por meio de atuadores, adaptar-se a mudanças, criar e perseguir metas. Os agentes são frequentemente referenciados pelo nome de bot, especialmente quando estão relacionados a jogos de computador ou videogames (RUSSELL, 2010).

Este trabalho visa criar um ambiente para competições entre agentes com o foco na facilidade de implementação destes, já que poderão ser implementados em qualquer linguagem de programação que permita a leitura e escrita de arquivos.

O trabalho está estruturado da seguinte forma. Nas subseções abaixo é mostrado os objetivos gerais e específicos e é explicado o contexto e a justificativa para o desenvolvimento deste trabalho. Na seção 2, as regras do jogo Risk são explicadas e exemplificadas com imagens. Na seção 3, o funcionamento de todo o sistema é explicado e exemplificado a partir de esquemas. Na seção 4, são apresentados alguns trabalhos relacionados, mostrando suas diferenças e semelhanças. A seção 5, explica a metodologia e o desenvolvimento de todo o trabalho, o desenvolvimento do jogo implementado em Python, a comunicação por arquivos *JSON* entre o jogo e os agentes, o funcionamento de 2 agentes simples de teste, alguns experimentos feitos com estes agentes e como funciona o visualizador. A seção 6 conclui o trabalho e a seção 7 demonstra algumas possibilidades de trabalhos futuros.

1.1 OBJETIVO GERAL

O objetivo geral deste trabalho é desenvolver um sistema capaz de proporcionar um ambiente para competições entre agentes, no jogo de tabuleiro Risk, que seja simples e fácil de utilizar.

O sistema é composto de duas partes, jogo e visualizador. O jogo é onde os agentes irão competir e o visualizador é onde o usuário poderá ter um feedback visual sobre partida entre os agentes.

1.1.1 Objetivo Específico

- Estudar e pesquisar as melhores ferramentas para a construção do sistema.
- Implementar o jogo de tabuleiro Risk, de maneira que agentes possam competir entre si.
- Permitir que qualquer linguagem de programação, com suporte a leitura e escrita de arquivos, possa ser utilizada para a implementação de agentes para o jogo.
- Implementar um visualizador de partidas, parte gráfica onde a partida jogada por agentes poderá ser visualizada como se fosse um vídeo.
- Como consequência do ambiente criado, incentivar a pesquisa e o estudo da área de IA.
- Criar agentes de teste, para demonstrar a utilização do sistema e suas possibilidades.

1.2 CONTEXTO E JUSTIFICATIVA

As salas de aula tradicionais, que são focadas no professor, caracterizam-se por um modelo de aprendizagem mais teórico onde a participação do aluno é mais passiva, os professores tem o papel apenas de transmitir um conhecimento e avaliar por meio de testes.

O benefício com essa abordagem tradicional é que o aluno pode aprender o básico da área da matéria e preencher as lacunas de conhecimento que qualquer iniciante possui, mas alunos que estão apenas começando podem ficar assustados com a quantidade e a profundidade das informações e é provável que perca a motivação rapidamente. Não é incomum no curso de ciência da computação ter alunos abandonando a faculdade por estes motivos (CARPIO et al., 2014b).

Como o livro (REEVE, 2018) explica, a motivação é influenciada por quatro fatores; contexto (ambiente e estímulos externos), temperamento (a condição interna de uma pessoa), objetivo (objetivo de comportamento e propósito) e instrumentos (instrumentos para atingir o objetivo). Para atingir seus objetivos, os seres humanos adquirem a motivação suficiente. Particularmente no que diz respeito aos alunos, a motivação para o desempenho acadêmico é de grande importância. Com essa motivação as pessoas são estimuladas a completar com sucesso uma tarefa, atingindo um objetivo ou um grau de aprendizagem (REEVE, 2018) (AMRAI et al., 2011).

E pensando nisso, utilizar apenas o modelo de aula tradicional, com um ambiente de aprendizado passivo, não é o melhor caminho, pois nem sempre estimulará os alunos (O'RIORDAN; KIRKLAND, 2008). E é nesse cenário que um modelo de aula mais moderno, centrados no aluno, é mais adequado, pois aprendizagem prática e teórica são aplicadas por um conhecimento flexível em vez de um convencional. A este respeito, os métodos interativos (competições e aulas práticas) permitem que os professores envolvam melhor os alunos (CARPIO et al., 2014b).

Os alunos também podem chegar com diferentes gostos, pensamentos e habilidades, influenciando assim, o seu grau de motivação. Pensando nessa diversidade, é necessário achar algo que abranja grande parte dos estudantes da área da computação, mais especifico a área de IA. É nesse contexto que entram os jogos, pois são utilizados para atrair a atenção dos estudantes, já que são na maioria dos casos os que levam as pessoas a desenvolver interesse na área de computação.

É pensando nesse modelo mais moderno de aprendizagem, juntamente com a utilização dos jogos, que competições de IA em jogos têm sido usadas com frequência e com sucesso, aumentando o desenvolvimento de pesquisas novas e inovadoras, já que competições motivam muitos estudantes e pesquisadores a trabalhar em problemas difíceis e também fornecem uma estrutura para contextualizar e comparar as pesquisas. Além disso, as competições podem ser usadas para apoiar o ensino de IA em geral, pois apresentam problemas muito concretos para os alunos abordarem e, normalmente, compartilharem abertamente os envios e resultados (SALTA; PRADA; MELO, 2021).

Dadas essas considerações, que o presente trabalho propõe a implementação e disponibilização de um sistema que possibilite, professores e estudantes, a fazer competições entre agentes para o jogo de tabuleiro Risk. Jogo este que foi escolhido por suas características estocásticas e simples e também por ser bastante conhecido pelos estudantes do país, para que possa atrair a maior quantidade de pessoas.

2 JOGO RISK

O Risk é um jogo de tabuleiro produzido pela Hasbro (HASBRO, 1993), que envolve sorte e estratégia, onde seu objetivo é dominar o mundo. Existem diferentes versões das regras em diversas edições lançadas pelo mundo. Este trabalho final de graduação utiliza uma adaptação mais simples das regras da edição de 1993 (HASBRO, 1993), lançada nos Estados Unidos.

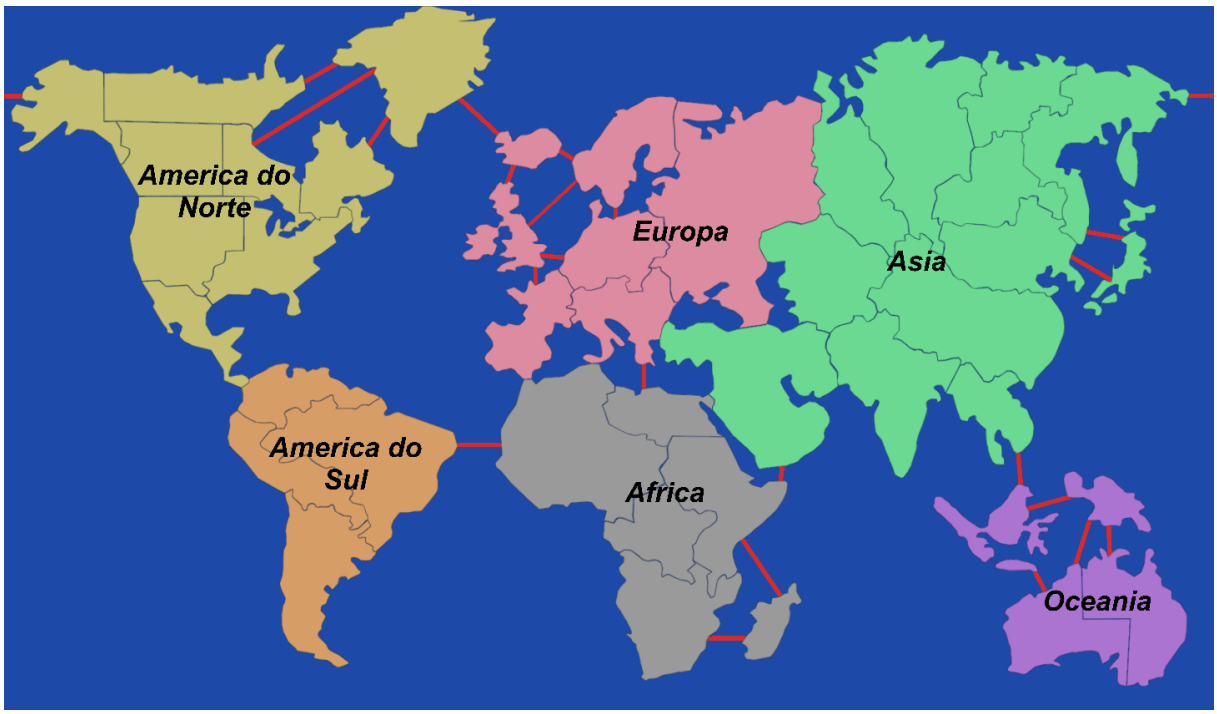
2.1 TABULEIRO

O tabuleiro do jogo é um mapa do mundo dividido em 6 continentes, composto por 42 países, que podem ser visualizados nas figuras 2.1 e 2.2.

Figura 2.1 – Mapa do jogo Risk especificando os 42 países

Fonte: Figura do autor

Figura 2.2 – Mapa do jogo Risk especificando os 6 continentes



Fonte: Figura do autor

2.2 REGRAS

É um jogo de estratégia por turnos, onde cada jogador só poderá jogar no seu turno e cada turno é divididos em 3 etapas: colocar novas tropas, atacar e fortificar.

2.2.1 Configuração Inicial

No início do jogo cada jogador receberá 21 países, distribuídos de forma aleatória. Após essa distribuição, cada jogador receberá 40 tropas, que serão colocadas também de forma aleatória entre os seus países.

2.2.2 Etapa de colocar novas tropas

Esta etapa acontece no início de cada turno, o jogador daquele turno recebe um certo número de tropas para colocar em qualquer país de seu domínio. O número de tropas recebidas é baseado na quantidade de países que o jogador domina e na quantidade de continentes que ele controla.

- **Países dominados:** para cada 3 países dominados o jogador recebe uma tropa nova, o mínimo que o jogador pode ganhar são 3 tropas.
- **Continentes dominados:** para cada continente que seja completamente dominado pelo jogador, ele recebe tropas bônus. A quantidade de bônus recebido por cada continente está relacionada ao tamanho de cada continente e pode ser visualizado na tabela 2.1.

Tabela 2.1 – Tropas extras para cada continente

Continente	Asia	América do Norte	Europa	Africa	Austrália	América do Sul
Tamanho	12 Países	9 Países	7 Países	6 Países	4 Países	4 Países
Número de tropas extras	+7	+5	+5	+3	+2	+2

Fonte: Adaptado de (HASBRO, 1993)

2.2.3 Etapa de Ataque

A segunda etapa do turno é a de ataque. Os jogadores podem atacar um país inimigo com qualquer país dominado que esteja adjacente. O jogador atacante escolhe o número de tropas para atacar, que pode ser de 1 a 3 tropas. O jogador deve deixar pelo menos uma tropa ocupando o país. Portanto, deve sempre haver 1 tropa a mais do que o número escolhido para atacar.

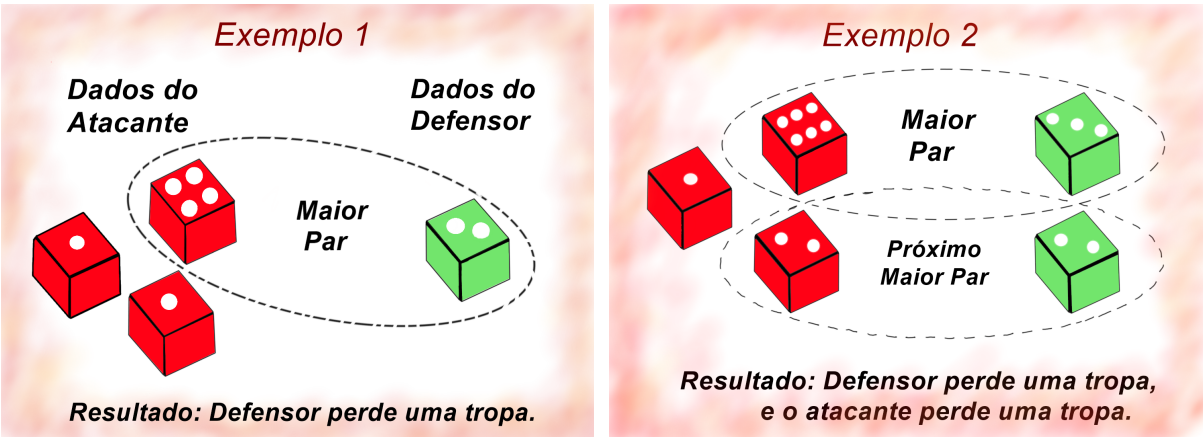
O ataque funciona jogando dados e comparando os seus valores com os dados do defensor. O número escolhido para o ataque é representa a quantidade de dados que serão jogados. O defensor pode lançar até no máximo 2 dados, cada um representando uma tropa.

Uma vez que todos os dados tenham sido lançados, o maior dado do atacante é comparado com o maior dado do defensor e o segundo maior dado do atacante é comparado com o segundo maior dado do defensor. O valor mais alto em cada comparação ganha, em caso de empate a vitória é do defensor. A tropa perdida em cada par é removida do tabuleiro. Exemplos visuais de como funciona o ataque podem ser vistos nas figuras 2.3 e 2.4. Um ataque pode terminar de três maneiras:

- O atacante decide encerrar a etapa de ataque (passar para a próxima etapa);
- O atacante fica sem tropas para atacar;
- O defensor perde todas as tropas.

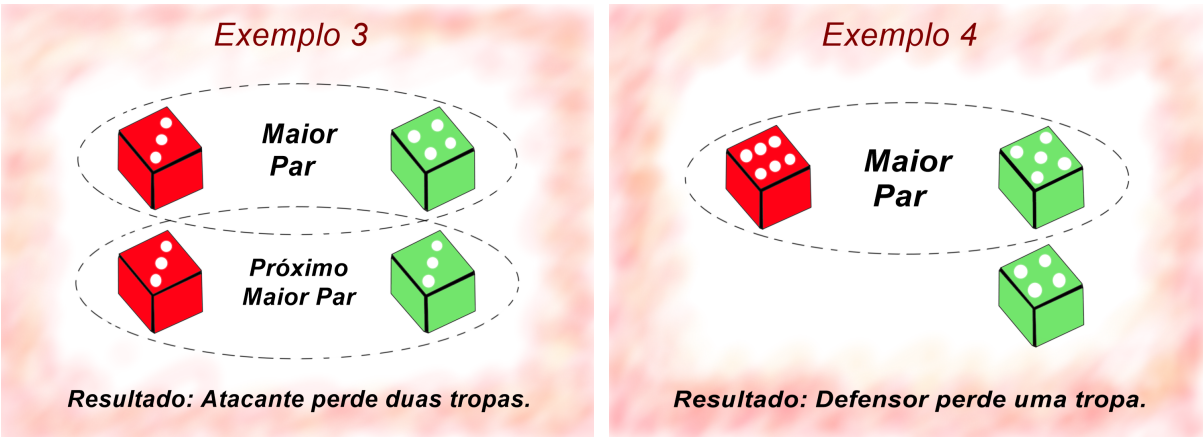
No terceiro caso, o atacante domina o país e deve obrigatoriamente mover o número de tropas que usou no ataque para o país atacado e em seguida pode mover mais quantas tropas quiser, sempre tendo que deixar pelo menos uma no seu país de origem.

Figura 2.3 – Exemplos de como o ataque funciona



Fonte: Figura do autor

Figura 2.4 – Exemplos de como o ataque funciona



Fonte: Figura do autor

2.2.4 Etapa de Fortificação

O terceira e última etapa do turno é a de fortificação, onde os jogadores podem fazer um único movimento com quantas tropas quiserem, entre seus países que tenham conexão, tendo que deixar pelo menos uma tropa no país de origem. Conexão significa ter um caminho entre o país que mandará as tropas e o que receberá entre os países

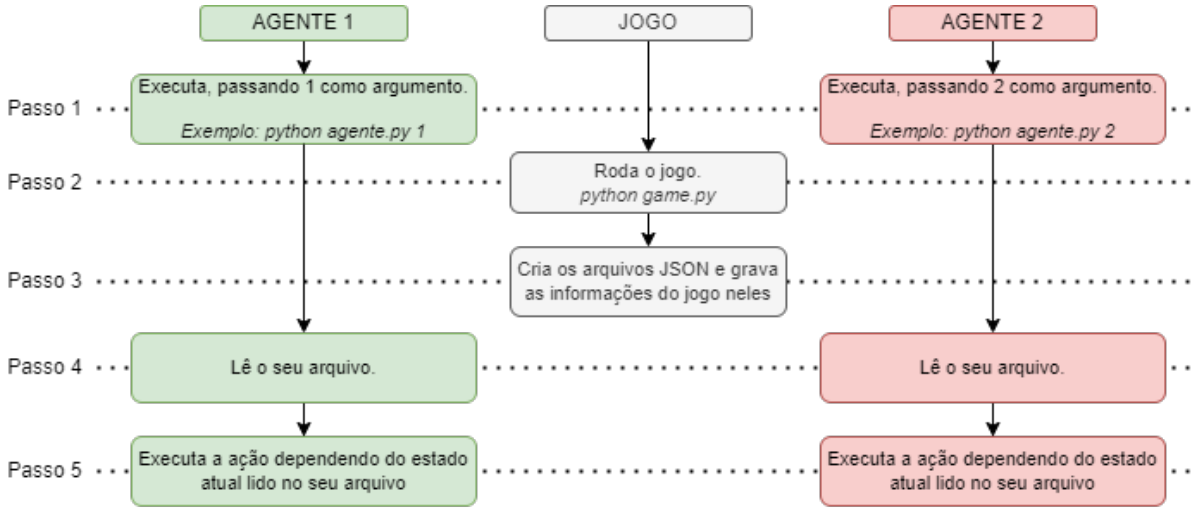
dominados do jogador. Quando finalizar a etapa de fortalecimento, o jogador passa o turno, iniciando o turno do próximo jogador.

Um jogador é eliminado quando não domina mais nenhum país no tabuleiro. O jogo termina quando apenas um jogador permanece.

3 FUNCIONAMENTO DO SISTEMA

O sistema implementado neste trabalho tem uma simples execução. Está dividido em 3 partes: inicial, execução e final. Estas que podem ser visualizadas nas figuras 3.1, 3.2 e 3.3, onde são mostrados a partir de esquemas o passo a passo para executar todo o sistema.

Figura 3.1 – Funcionamento da parte inicial do sistema

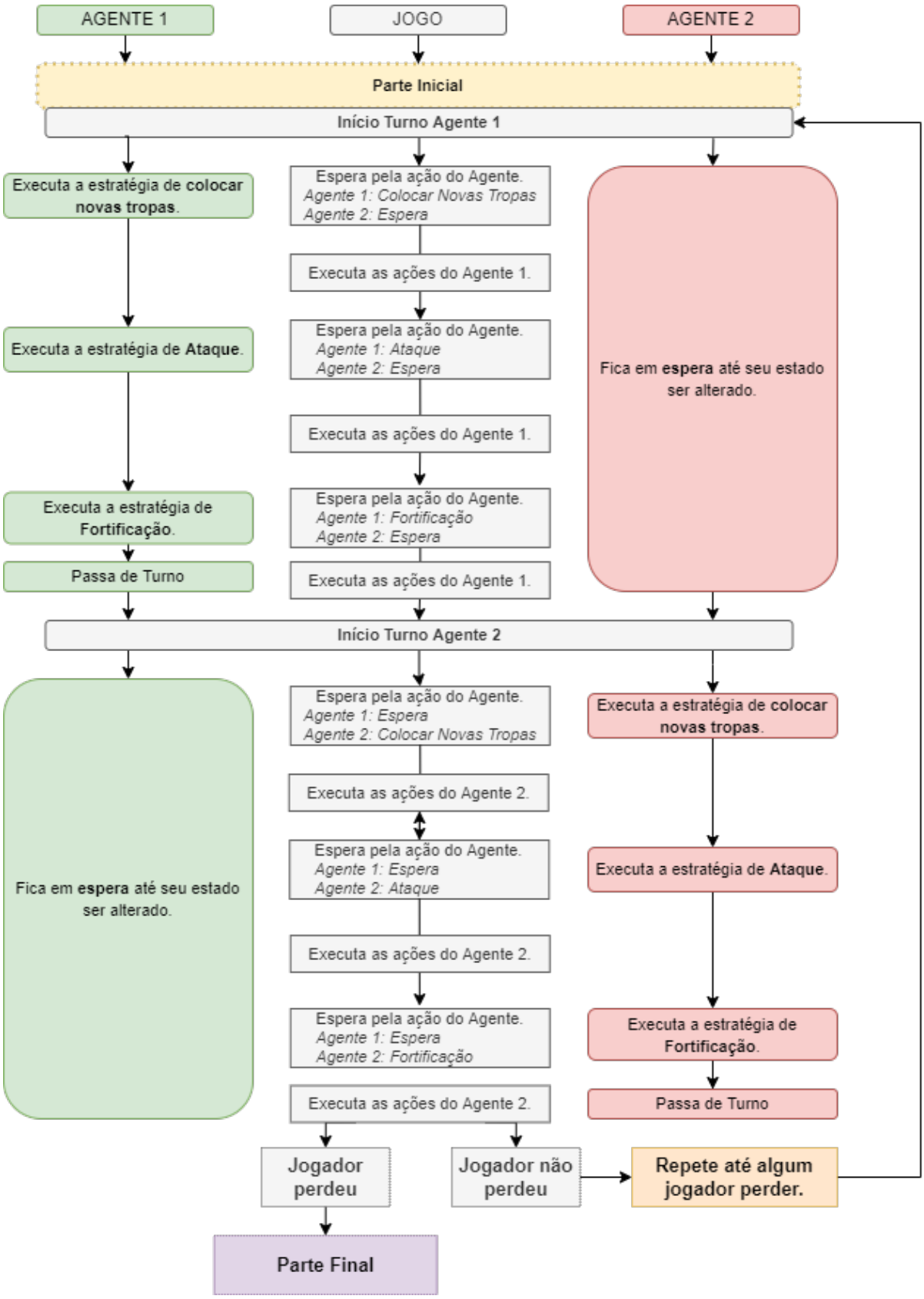


Fonte: Figura do autor

A figura 3.1, mostra o esquema da parte inicial do sistema, onde o jogo e os agentes são inicializados. Cada um é um programa diferente e são inicializados separadamente, primeiro são executados os dois agentes e depois o jogo, que cria dois arquivos para cada jogador e os preenche com as configurações iniciais. Assim os agentes podem ler seus respectivos arquivos e já saber o que fazer.

Após a primeira leitura dos arquivos pelos agentes é que começa a parte de execução do sistema, que pode ser visualizado no esquema da figura 3.2. Onde os agentes a partir do estado informado a eles pelo seus arquivos, podem começar a executar suas estratégias e mandar comandos para o jogo através do arquivo de comandos. Com isso o jogo vai executar estes comandos e atualizar as informações nos arquivos dos agentes.

Figura 3.2 – Funcionamento da parte de execução dos agentes no sistema

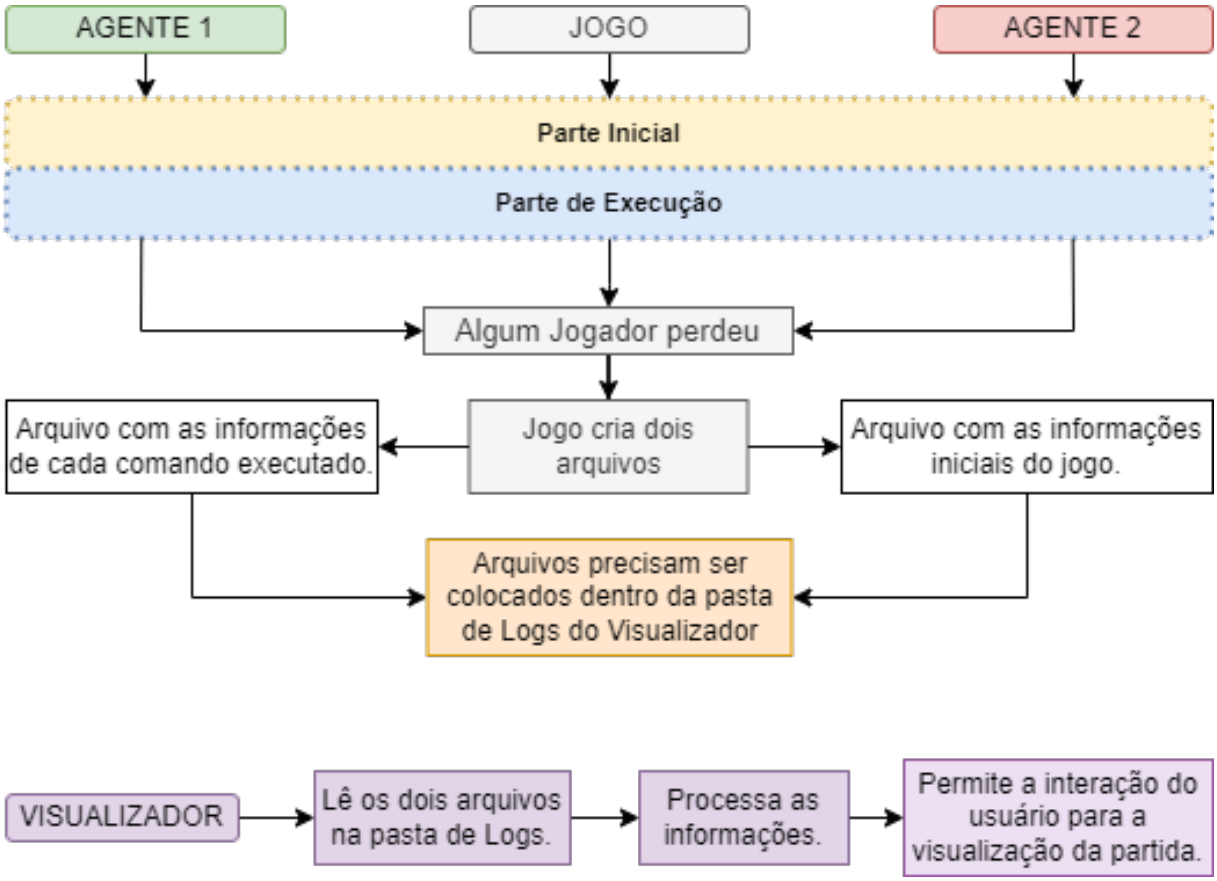


Fonte: Figura do autor

Quando algum agente perder, ou seja, não tiver mais países dominados, o jogo termina e vai para a última parte. Onde são criados dois arquivos com as informações iniciais do jogo e com as informações de cada comando feito ao longo da partida inteira.

Então será necessário que este dois arquivos sejam colocados manualmente pelo usuário dentro da pasta *Logs* que está dentro dos arquivos do visualizador. Após colocados na pasta, só será necessário executar o programa do visualizador na Unity, para que assim possa ver toda a partida como se fosse um filme, podendo escolher qualquer momento da partida para visualizar.

Figura 3.3 – Funcionamento da parte final de visualização do sistema



Fonte: Figura do autor

4 TRABALHOS RELACIONADOS

- **ColorShapeLinks: A board game AI competition for educators and students (FACHADA, 2021):** ColorShapeLinks é um framework de competição de jogos de tabuleiro para IA. O framework é baseado em uma versão do jogo de tabuleiro Simplexity. Ele oferece interfaces gráficas e baseadas em texto, além de ser um framework de desenvolvimento aberto e documentado. ColorShapeLinks oferece front-ends de console Unity e .NET. Os agentes são implementados em C# e executados sem modificações em qualquer um dos front-ends fornecidos.

  Este trabalho não oferece uma usabilidade muito grande, pois se limita a linguagem de programação C# para a implementação de seus agentes. A complexidade do jogo de tabuleiro Simplexity é baixa, já que o jogo tem um número máximo de turnos, jogadas específicas e sem nenhum fator aleatório. Dessa forma soluções ótimas tendem a aparecer no curto e médio prazo, diminuindo a vida útil do programa em questão de novas abordagens de IA.

- **Blood Bowl: A New Board Game Challenge and Competition for AI (JUSTESEN et al., 2019):** as implementações existentes do jogo Blood Bowl eram de código fechado e não tinham uma interface para IAs. Então os autores deste trabalho desenvolveram, em Python, o próprio motor de jogo chamado Fantasy Football AI (FFAI). O FFAI também vem com um aplicativo da web simples que implementa uma interface de usuário para humanos jogarem contra outros humanos, online ou local, ou contra bots.

  O desenvolvimento de agentes para este trabalho se limita a linguagem de programação Python, o que diminui a sua usabilidade. O jogo de tabuleiro Blood Bowl não é muito conhecido no Brasil. É um jogo que não tem número fixo de turnos e é estocástico, o que faz com que tenha fatores aleatórios e por isso é considerado mais complexo para se desenvolver agentes.

- **Lux Delux (Sillysoft, 2002):** Lux Delux é uma implementação em java do jogo de tabuleiro Risk, feita pela empresa Sillysoft, que disponibiliza um SDK para implementar IAs ou gerar mapas para o Lux Delux. É disponibilizado também diversas implementações de IAs com diferentes estratégias para o jogo. É uma versão paga do jogo Risk que se limita a implementação de agentes na linguagem de programação Java.

- **An implementation of the 7 Wonders board game for AI-based players (BETTKER et al., 2020):** o trabalho descreve uma implementação do jogo de tabuleiro 7 Wonders com o objetivo de permitir a criação, aprimoramento e o teste de agentes de IA. É implementado em C++, com capacidade de rodar centenas de jogos por

minuto usando arquivos *JSON* simples como entrada e saída, aceitando qualquer linguagem.

- **Open classroom: enhancing student achievement on artificial intelligence through an international online competition (CARPIO et al., 2014a):** este artigo questiona as limitações da aprendizagem formal (por exemplo, comunicação unidirecional, metodologia rígida, abordagem orientada para resultados) e como ela pode influenciar na motivação dos estudantes e consequentemente reduzir o progresso acadêmico.

  Então para tornar os processos de aprendizagem mais divertidos e motivadores, o trabalho apresenta uma experiencia educacional desenvolvida por alunos do curso de Ciência da Computação da Universidade de Huelva (Espanha). Esta experiencia consistia em participar de uma competição, chamada Google AI Challenge, com dois grupos de alunos durante um semestre inteiro. Eles chegaram a conclusão, a partir da opinião e do conhecimento dos alunos participantes, que essas competições permitiram aos alunos consolidar conceitos teóricos, melhorar a percepção sobre os cursos, promover motivação e interesse.

- **A Game AI Competition to foster Collaborative AI research and development (CHESANI et al., 2017):** neste artigo os autores relatam o ensino de IA aplicando a abordagem experiencial, chamada “aprender fazendo”, onde o ensino tradicional e formal é integrado a uma atividade prática (competição de jogos). Onde estudantes foram desafiados a desenvolver agentes para o jogo Nine Men’s Morris.

  Para avaliar o impacto dessa abordagem os estudantes participantes responderam questionários sobre aspectos técnicos e sobre a sua opinião sobre a competição. E com base nas respostas destes questionários consideraram a abordagem bem sucedida. os estudantes confirmaram a melhoria de seus conhecimentos e declararam um interesse maior pelos tópicos do curso e pela IA em geral.

- **Uma comparação entre abordagens de IA para o jogo Risk (FERRARI, 2022):** no trabalho de F. René são desenvolvidos agentes utilizando o sistema apresentado nesta monografia. São três agentes heurísticos distintos e um agente racional, todos implementados na linguagem de programação Python. Os quatro agentes são testados e comparados entre si.

Estudando e Visualizando as características de cada um destes trabalhos, que o sistema desenvolvido neste projeto de TCC foi pensado. Ele permite competições entre IAs no jogo de tabuleiro Risk. Jogo que se tornou bastante conhecido no Brasil, pois teve uma versão brasileira, lançada pela Grow em 1971, chamada War, sendo assim mais chamativo para seus residentes.

O sistema tem a finalidade de aproximar novos estudantes para a área de IA e de lhes passar um ambiente mais livre e fácil. Para alcançar esses objetivos, três pontos foram desenvolvidos de maneira diferente dos trabalhos observados, que são: a complexidade do jogo, como será distribuído e como poderão ser implementados os agentes para o jogo.

No primeiro ponto, diferentemente do (FACHADA, 2021), o jogo Risk tem características mais complexas, pois ele não tem um número fixo de turnos e é estocástico. Sendo assim um jogo que pode ter agentes simples e complexos e com mais possibilidades futuras de implementação.

No segundo ponto, diferentemente do (Sillysoft, 2002), todo o conteúdo será disponibilizado de forma gratuita e aberta, incluindo o jogo Risk desenvolvido em Python, dois agentes de exemplo e um visualizador gráfico das partidas.

No terceiro ponto, foi utilizado a ideia de comunicação, entre jogo e agentes, do trabalho (BETTKER et al., 2020), que é feita através de arquivos *JSON*, permitindo a implementação de agentes em qualquer linguagem de programação que permita leitura e escrita nesse tipo de arquivo. O que difere dos trabalhos (FACHADA, 2021), (Sillysoft, 2002) e (JUSTESEN et al., 2019) que apenas permitem a implementação em uma única linguagem.

5 METODOLOGIA E DESENVOLVIMENTO

Após pesquisas sobre competições e jogos com o foco em IA, não foi encontrado implementações que permitissem a criação de agentes de maneira simples e com suporte a diferentes linguagens de programação, principalmente de jogos com características estocásticas e com visão completa (jogadores tem todas as informações sobre o inimigo e do tabuleiro). Portanto, planejou-se o desenvolvimento do jogo de tabuleiro Risk focado em competições entre IAs e na facilidade e liberdade de implementação de seus agentes.

Este trabalho está dividido em 3 partes. A primeira refere-se ao desenvolvimento do jogo e sua API, que utilizarão o tipo de arquivo *JSON* para a troca de informação entre jogadores e jogo juntamente com a linguagem de programação Python, pois não só oferece grande suporte a leitura e escrita de arquivos no formato *JSON*, como também é muito utilizada para diversas implementações de IAs.

A segunda é destinada à implementação de agentes simples, que usam a API desenvolvida na primeira parte. Para demonstrar e testar a API foram implementados 2 agentes simples com diferentes estratégias.

Já a terceira e última parte é focada na visualização, onde primeiramente foi necessário implementar um sistema no jogo para gravar as partidas jogadas em arquivos que sejam legíveis por humanos, facilitando a manutenção e detecção de problemas. A partir disso, foi possível desenvolver a parte gráfica na plataforma de desenvolvimento 3D Unity, onde são lidos os arquivos com as partidas armazenadas e mostrado de forma visual todo o andamento dela. Todas as partes estão disponíveis em (Pavin, Thiago and Gargano Ferrari, René, 2022).

5.1 PROJETO E DESENVOLVIMENTO DO JOGO

Explicadas abaixo cada classe do jogo, que podem ser visualizadas na figura 5.1.

- **Classe *Country*:** serve para guardar todas as informações do país, como o seu nome, lista de países que fazem fronteira (vizinhos), qual jogador o domina e o número de tropas que estão no país. Contém a função ‘add neighbors’, que adiciona os países que fazem fronteira para a sua lista de vizinhos.

- **Classe Continent:** contém todas as informações do continente, como seu nome, uma lista com todos os países que pertencem a ele, se tem algum jogador que domina todos os países dele e quantas tropas extra ele dá para cada jogador quando dominado por inteiro.

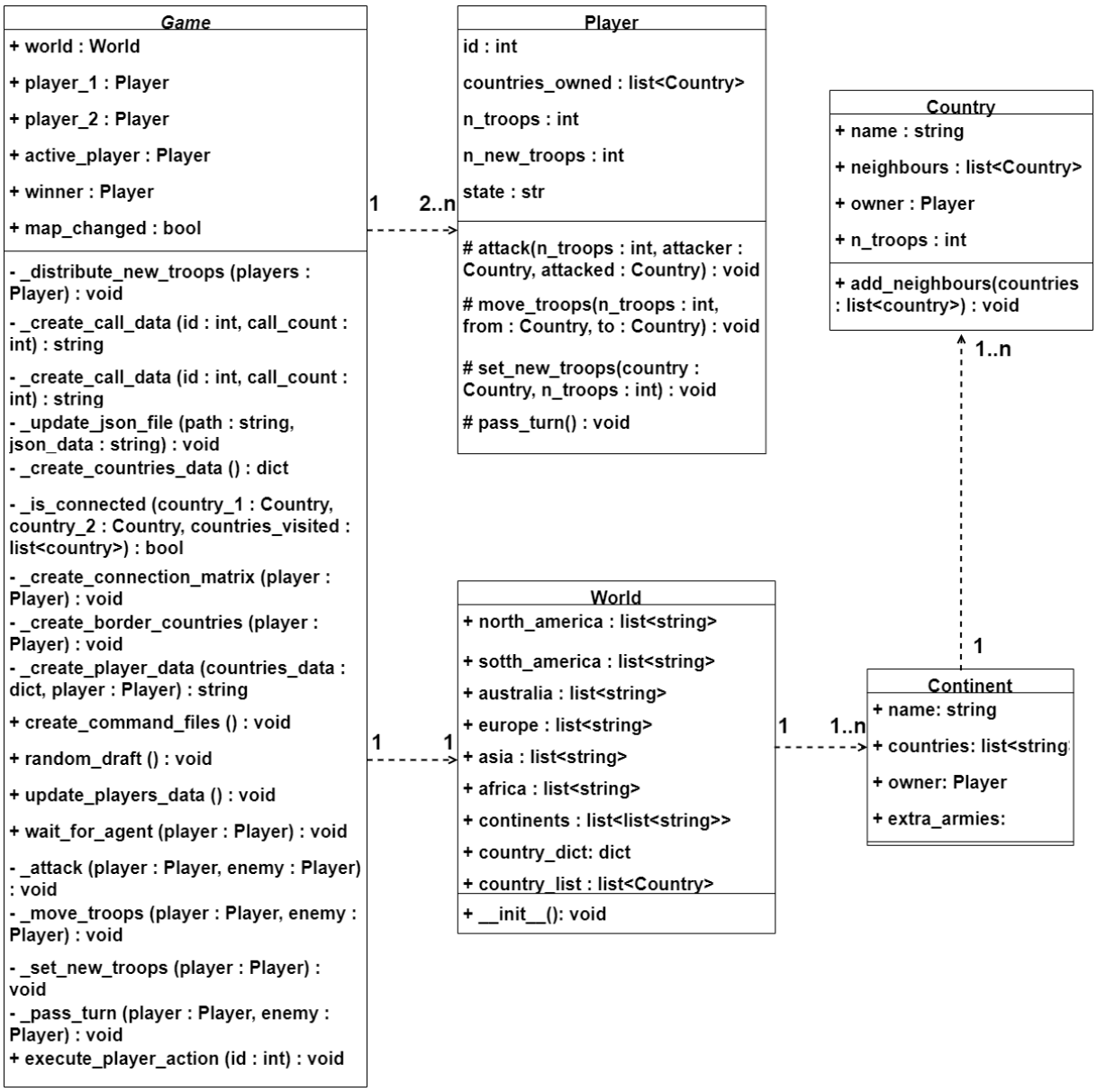
- **Classe *World*:** é responsável por criar todo o mapa. Cria cada continente e cada país e organiza as suas informações, para poder fazer todas as conexões entre países e continentes. Criando no final um dicionário com todos os países que utiliza o nome de cada um como sua chave de busca, para facilitar a busca quando algum jogador precisar destas informações.
- **Classe *Game*:** classe principal, responsável por gerenciar todo o fluxo do jogo e guardar todas as informações do jogo, como o mapa, os jogadores, a etapa em que cada jogador está, contador de chamadas, o caminho para os arquivos de informações e os arquivos de chamadas e uma flag que sinaliza se algo foi modificado no mapa. Esta classe contém as seguintes funções:
  - **distribute new *troops*:** cada turno de 'mobilizing' calcula a quantidade de tropas que o jogador deve receber, a partir da quantidade de países que domina.
  - **create call data:** cria o dicionário modelo para cada jogador, para permitir a criação de chamadas pelos jogadores. Retorna uma 'string' com o dicionário serializável para o formato *JSON*.
  - **update *JSON* file:** recebe a 'string' com o dicionário serializável para o formato *JSON* das informações para o jogador e atualiza o arquivo *JSON* que contém as informações de cada jogador.
  - **create countries data:** cria o dicionário contendo as informações de todos os países, para facilitar a leitura e busca dos jogadores.
  - **is connected** retorna se dois países têm caminho entre eles pelos seus países dominados.
  - **create connection matrix:** cria uma matriz com todas as possíveis conexões (se existe caminho) entre os países dominados.
  - **create border countries:** atualiza os países inimigos que cada país dominado faz fronteira.
  - **create player data:** cria o dicionário que guarda todas as informações necessárias para cada jogador. Retorna uma 'string' com o dicionário serializável para o formato *JSON*.
  - **create command files:** cria os arquivos de chamadas e atualiza com o modelo recebido, guardando o tempo que foi criado.
  - **random draft:** distribui aleatoriamente os países entre os jogadores, 21 para cada. Depois distribui aleatoriamente as tropas iniciais aleatoriamente nos países de cada jogador.
  - **update players data:** atualiza os arquivos com as mudanças feitas naquele turno

  - **wait for agent:** entra em etapa de espera até o jogador ativo alterar o arquivo com alguma nova chamada.
  - **execute player action:** lê a chamada do jogador e executa o comando que estiver nela, pode ser um comando de ataque, mover tropas, colocar novas tropas ou de passar o turno.

- **Classe Player:** contém todas as informações do jogador, como um contador de chamada, seu 'ID', lista de países que domina, quantidade de tropas novas recebidas, total de tropas somando todos os seus países, qual a etapa que o jogador está, guarda a matriz de conexão, dicionário com os países inimigos que fazem fronteira com cada país e uma instância da classe *control*. A sua função é de controlar a execução das ações do jogador.

- **Classe Control:** contém algumas informações de controle de cada jogador, como o último tempo em que seu arquivo foi alterado, a quantidade de chamadas feitas, os caminhos para os arquivos de dados e de chamadas e a informação do último comando feito.

Figura 5.1 – Arquitetura mostrando os métodos do projeto do jogo



Fonte: Próprio autor.

O desenvolvimento começou pelas classes que definiam os países e mundo do jogo, *Country* e *World*. Pensando na facilidade de acesso aos dados dos países e suas ligações foi utilizado a estrutura de dados *dicionário*, onde a chave é o nome do país. Foi necessário o uso de um grafo para representar o mapa. Com a implementação da função *add neighbours* foi possível definir as arestas do grafo (conexões entre países).

Para iniciar a construção da lógica do jogo, foi desenvolvido a classe dos *jogadores*, para assim poder ser definido suas ações e informações guardadas. Após, criou-se a classe principal *Game*, onde toda a lógica do jogo pode ser implementada, da administra-

ção do mapa até a comunicação entre jogo e agentes.

No jogo Risk original existem três jogadores, dois que são jogadores normais, no nosso caso agentes, e um terceiro neutro, que apenas ocupa alguns países aleatórios e os defende, sem atacar os outros jogadores. Mas depois de alguns testes, foi entendido que esse jogador neutro apenas traria complexidade a implementação sem nenhum retorno positivo para os agentes, por isso na implementação do Risk para este trabalho, deixou-se apenas os dois jogadores controlados por agentes dividirem o mapa.

No jogo, existe a possibilidade de movimentação de tropas de um país dominado para outro. Só é permitido esta movimentação na etapa de fortificação e quando um país é dominado, entre dois países conectados. Então, para a representação das tropas, somente é necessário guardar em cada país o seu número de tropas com uma variável 'inteira', para quando for movida diminuir ou aumentar esse número.

Para a comunicação entre o jogo e os agentes (jogadores), é utilizado arquivos *JSONs*, onde o jogo cria dois arquivos para cada jogador, um para as informações dos países e do jogador e outro para receber os comandos (ações) dos jogadores.

Visto que o jogo tem que ficar atento a qualquer modificação nos arquivos *JSONs* para verificar se algum jogador mandou algum comando novo, foram pensados em algumas alternativas mais eficientes, que não fugissem de um dos objetivos do trabalho, que é facilitar a implementação de agentes e entregar maneiras simples para cada agente ter liberdade de ser implementado como quiser e em qualquer linguagem com suporte a leitura e escrita a arquivos. Por isso, foi selecionada a biblioteca *os*, que pode fazer uma operação de verificação de arquivos com apenas uma linha de código na maioria das linguagens.

Para facilitar a implementação dos agentes algumas funções auxiliares para gerar informações importantes foram implementadas, na classe *Game*, como verificar a existência de conexão entre cada país dominado e verificar uma lista com todos os países inimigos que cada país dominado tem fronteira. Dessa maneira não é deixado todo o trabalho para os agentes, facilitando sua implementação. Esses dados são entregues como dicionários juntamente com as informações dos arquivos *JSONs*.

Como este jogo tem o objetivo de ser usado em competições, é necessário ter algumas seguranças contra trapaças. A primeira delas é o jogo manter os dados do mapa e tropas dentro do jogo, então mesmo que algum agente tente mudar os arquivos o jogo não vai ser alterado. A segunda é sobre os comandos feitos pelos agentes, o jogo verifica se o comando é possível de ser executado e caso o agente esteja querendo mandar algum comando com dados falsos o jogo irá retornar um log com os erros encontrados.

5.1.1 Arquivos *JSONs*

Para possibilitar a liberdade de implementação dos agentes em qualquer linguagem de programação, a comunicação entre o jogo e os agentes utiliza arquivos *JSONs*. Arquivos que no início de cada partida serão criados pelo jogo. Cada jogador terá 2 arquivos *JSON*, arquivo do jogador e arquivo dos comandos.

O **arquivo dos jogadores** é onde o jogo vai colocar todas as informações que são relevantes para os agentes jogar e tomar decisões. As informações armazenadas nele serão atualizadas toda a vez que o estado do jogo modificar. Nele são guardadas as seguintes informações:

- **count:** contador armazenando quantas vezes o arquivo já foi atualizado. Serve para ter uma sincronização com os jogadores mais precisa.
- **id:** identificador (1 ou 2) de qual jogador pertence os dados.
- **n new troops:** número identificando quantas tropas novas o jogador recebeu para serem utilizadas na etapa de colocar novas tropas.
- **n total troops:** número identificando o total de tropas que jogador possui.
- **enemy n total troops:** número identificando o total de tropas que jogador inimigo possui.
- **state:** etapa (atacando, conquistando, fortificando, colocando novas tropas ou esperando) em que jogador está no momento.
- **countries owned:** lista com o nome de todos os países que o jogador domina.
- **countries data:** informações sobre cada país do mapa (nome, lista com países vizinhos, identificador de qual jogador o domina e o número de tropas presentes no país).
- **border countries:** lista com os países dominados que tem inimigos como vizinhos e uma lista com os nomes desses vizinhos.
- **connection matrix:** matriz de conexão entre os países dominados, para facil acesso a qual país é possível mover tropas.

Abaixo segue o modelo do arquivo JSON dos jogadores, mostrando alguns exemplos e como o as informações são organizadas e visualizadas no arquivo:

```
Modelo de arquivo JSON dos jogadores:
{
```

```
    "count": 0,
    "id": 1,
    "n_new_troops": 0,
    "n_total_troops": 0,
    "enemy_n_total_troops": 0,
    "state": "attacking || conquering || fortifying || mobilizing || waiting",
    "countries_owned": ["name", "name", "..." ,"name"],
    "countries_data": {
        "country_1": {
            "name": "name",
            "neighbours": ["name", "name"],
            "owner": 1,
            "n_troops": 1},
        "country_2": {
            "name": "name",
            "neighbours": ["name", "name"],
            "owner": 1,
            "n_troops": 1},
        "country_3": {
            "name": "name",
            "neighbours": ["name", "name"],
            "owner": 2,
            "n_troops": 1}
    },
    "border_countries": {
        "Country_1": ["Country_3"],
        "Country_2": ["Country_3"],
    },
    "connection_matrix": {
        "Country_1": {
            "Country_2": true,
            "Country_3": false},
        "Country_2": {
            "Country_1": true,
            "Country_3": false}
}
```

O **arquivo dos comandos** é onde o jogador coloca as informações sobre o próximo comando que quer realizar. Toda a vez que o jogo verificar uma mudança nesse arquivo,

executará o comando a partir desse arquivo (cada jogador tem o seu arquivo). Nele são guardadas as seguintes informações:

- **id:** identificador (1 ou 2) de qual jogador pertence os dados.
- **count:** contador armazenando quantas vezes o arquivo já foi atualizado, serve para ter uma sincronização com os jogadores mais precisa.
- **command:** onde as informações específicas do comando são armazenadas, cada comando possível tem uma configuração diferente:
  - **ataque:** nome (attack) e argumentos (número de dados, país atacante e país atacado).
  - **mover tropas:** nome (move troops) e argumentos (número de tropas, país origem e país destino).
  - **colocar novas tropas:** nome (set new troops) e argumentos (número de tropas e país destino).
  - **passar o turno:** nome (pass turn).

Abaixo segue o modelo do arquivo JSON dos comandos, mostrando alguns exemplos e como o as informações são organizadas e visualizadas no arquivo:

Modelo de arquivo JSON dos comandos:

```
{
    "id": 1,
    "count": 0,
    "command_1": {
        "name": "attack",
        "args": ["n_dice", "attacker", "attacked"]},
    "command_2": {
        "name": "move_troops",
        "args": ["n_troops", "from_country", "to_country"]},
    "command_3": {
        "name": "set_new_troops",
        "args": ["n_troops", "country_name"]},
    "command_4": {
        "name": "pass_turn",
        "args": []}
}
```

5.2 DESENVOLVIMENTO DOS AGENTES

Foram desenvolvidos 2 agentes com heurísticas diferentes. Este agentes apenas foram desenvolvidos para testar o funcionamento do sistema, para demonstrar como utilizar o sistema em geral e ter alguns exemplos para que os usuários possam se basear. Os agentes são programas diferentes, separados do jogo, que podem ser implementados em qualquer linguagem. Os agentes implementados neste trabalho foram desenvolvido em Python e para executa-los é preciso passar um valor por parâmetro, que é o id (1 ou 2) do jogador. A partir desse valor o agente vai saber qual o caminho para os arquivos *JSON* que ele deve monitorar e alterar.

E com a leitura dos arquivos o agente vai saber em qual etapa ele está e quais são as informações gerais do jogo e assim executar a lógica para decidir qual o próximo comando a ser passado para o jogo. Cada etapa no agente tem sua estratégia específica, nestes 2 agentes teste foram utilizadas estratégias bastante simples. Estão disponíveis agentes mais complexos em (Gargano Ferrari, René, 2022), agentes estes explicados e desenvolvidos no trabalho (FERRARI, 2022).

A estratégia de ataque dos agentes implementados neste trabalho, utilizam as probabilidades de vitória e derrota para cada configuração de dados, atacante e defensor, uma configuração de exemplo seria, se o Atacante (A) joga 1 dado e o Defensor joga também 1 dado, a probabilidade do Atacante perder 1 tropa é de 58.3% e a probabilidade do Defensor perder 1 tropa é de 41.7%. As outras configurações de dados possíveis e suas probabilidades podem ser visualizadas na tabela, 5.1.

Tabela 5.1 – Probabilidades dos resultados de ataque. A e D são abreviações para Atacante e Defensor respectivamente.

Tropas		**Resultados**				
A	**D**	**A Perde 2**	**A Perde 1**	**Ambos Perdem 1**	**D Perde 1**	**D Perde 2**
1	1	-	58.3 %	-	41.7 %	-
1	2	-	74.5 %	-	25.5 %	-
2	1	-	42.1 %	-	57.9 %	-
2	2	44.8 %	-	32.4 %	-	22.8 %
3	1	-	34 %	-	66 %	-
3	2	29.3 %	-	33.6 %	-	37.1 %

Fonte: Adaptado de (WOLF, 2005)

Abaixo são explicados dois agentes diferentes, um se chama **Agente Simples** e o outro **Agente Simples Randômico**. Será explicada a lógica de cada uma de suas etapas.

5.2.1 Agente Simples

- **colocar novas tropas:** para esta etapa o agente pega a lista de países dominados que tem vizinhos inimigos e vai colocando 1 tropa em cada, até as tropas novas acabarem. Estratégia simples de defesa das bordas, sem pensar na quantidade de tropas que cada país inimigo tem.
- **ataque:** para a etapa de ataque o agente pega a lista de países dominados que tem vizinhos inimigos e faz algumas verificações a partir das probabilidades de ganho e perda, que pode ser visualizadas na tabela 5.1, para saber se é válido atacar cada país dessa lista. Toda a vez que a probabilidade de ganhar é grande ele escolhe atacar, colocando no arquivo de comandos os argumentos para realizar o ataque.
- **fortificação:** para a etapa de fortificação o agente vai verificar qual país dominado que não tem vizinho inimigo com mais tropas e passar todas as suas tropas (deixando apenas uma, regra do jogo) para o país dominado que tiver o maior número de inimigos como vizinhos, sempre verificando se existe conexão entre estes 2 países. Caso não achar algum país que tenha conexão, passa o turno.

5.2.2 Agente Simples Randômico

Este agente é uma versão do *Agente Simples*, 5.2.1, com algumas decisões sendo tomadas com fatores aleatórios.

- **colocar novas tropas:** para esta etapa o agente pega uma países aleatórios da lista de países dominados que tem vizinhos inimigos e vai colocando 1 tropa em cada, até as tropas novas acabarem.
- **ataque:** utiliza as mesmas verificações que o *Agente Simples*, adicionando um fator de decisão aleatório, cada vez que puder atacar terá 80% de chance de atacar e 20% de passar a vez.
- **fortificação:** para a etapa de fortificação o agente vai pegar um país dominado aleatório e mandar todas as suas tropas para algum país aleatório que tenha inimigos como vizinhos.

5.2.3 Experimentos

Para poder testar o funcionamento do jogo e avaliar o desempenho dos agentes, foram feitos alguns experimentos. Nestes experimentos foram avaliados a taxa de vitória, o tempo de execução da partida e a quantidade de escritas nos arquivos pelo jogo, agente 1 e agente 2.

De início apenas um cenário seria testado, *Agente Simples* (5.2.1) contra o *Agente Simples Randômico* (5.2.2), mas durante os experimentos foi notado que dependendo da estratégia utilizada pelos agentes, eles poderiam ter vantagem sendo os primeiros a jogar.

Portanto, foram necessários alguns outros cenários de teste, alterando a ordem de qual agente começa jogando, para ter mais precisão nas avaliações. Cada cenário foi rodado 50 vezes, os cenários foram os seguintes:

Tabela 5.2 – Resultados dos testes entre dois Agente Simples

Porcentagem de Vítória (Jogador 1)	**Porcentagem de Vítória (Jogador 2)**	**Média de Escrita em Arquivo (Jogo)**
75,4 %	24,6 %	720
Média de Escrita em Arquivo (Jogador 1)	**Média de Escrita em Arquivo (Jogador 2)**	**Tempo Médio de Execução**
217	141	2,661459287

Fonte: Próprio autor.

Tabela 5.3 – Resultados dos testes entre Agente Simples como jogador 1 e Agente Simples Randômico como jogador 2

Porcentagem de Vítória (Jogador 1)	**Porcentagem de Vítória (Jogador 2)**	**Média de Escrita em Arquivo (Jogo)**
100 %	0 %	627
Média de Escrita em Arquivo (Jogador 1)	**Média de Escrita em Arquivo (Jogador 2)**	**Tempo Médio de Execução**
233	80	2,266431395

Fonte: Próprio autor.

Tabela 5.4 – Resultados dos testes entre Agente Simples Randômico como jogador 1 e Agente Simples como jogador 2

Porcentagem de Vitória (Jogador 1)	**Porcentagem de Vitória (Jogador 2)**	**Média de Escrita em Arquivo (Jogo)**
14 %	86 %	834
Média de Escrita em Arquivo (Jogador 1)	**Média de Escrita em Arquivo (Jogador 2)**	**Tempo Médio de Execução**
164	253	3,077085388

Fonte: Próprio autor.

Tabela 5.5 – Resultados dos testes entre 2 Agente Simples Randômico

Porcentagem de Vitória (Jogador 1)	**Porcentagem de Vitória (Jogador 2)**	**Média de Escrita em Arquivo (Jogo)**
64 %	36 %	2372
Média de Escrita em Arquivo (Jogador 1)	**Média de Escrita em Arquivo (Jogador 2)**	**Tempo Médio de Execução**
650	536	7,938946028

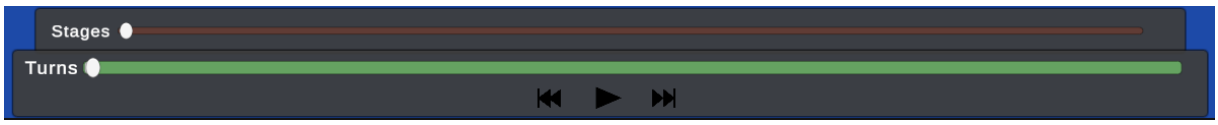
Fonte: Próprio autor.

5.3 DESENVOLVIMENTO DO VISUALIZADOR

Foi desenvolvido o visualizador de partidas na plataforma Unity. Este programa serve para os estudantes e professores terem um feedback visual de como os seus agentes estão se comportando em cada partida jogada. Para dessa forma entender em qual parte o seu agente está ganhando ou perdendo, para assim tomar decisão sobre o seu desenvolvimento.

Este visualizador funciona a partir da leitura de 2 arquivos *JSONs* de logs da partida, gerados pelo jogo no final da partida, nesses arquivos serão salvos todos os comandos feitos pelos jogadores e suas informações e o estado inicial do jogo, para que a partir dessas informações possa ir alterando dependendo de qual turno ou comando for selecionado. Para selecionar os turnos ou comandos, existem 2 sliders de controle, assim a pessoa que estiver utilizando pode arrastar este slider e ir alternando entre os turnos e comandos da partida, estes sliders podem ser vistos na figura 5.2. Abaixo destes dois sliders tem 3 botões de controle, que podem avançar um turno, voltar um turno e ativar ou desativar o andamento automático dos turnos e comandos.

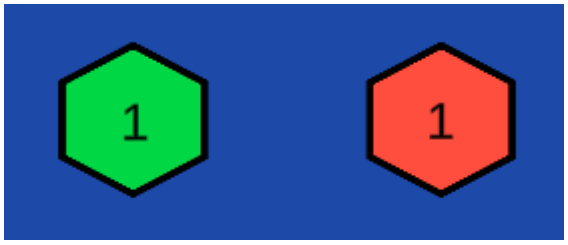
Figura 5.2 – Sliders de controle de turno e comando



Fonte: Próprio autor.

Para a visualização dos estados do jogo um mapa foi desenhado e nele cada país tem um hexágono que mostram a cor do jogador que domina aquele país e o número de tropas que ele possui. A cor do jogador 1 é a verde e a do jogador 2 é a vermelha. Um exemplo dos hexágonos pode ser visualizado na figura 5.3 e uma visão completa de como o mapa é mostrado pode ser visualizado na figura 5.4.

Figura 5.3 – Exemplo de estado de cada país

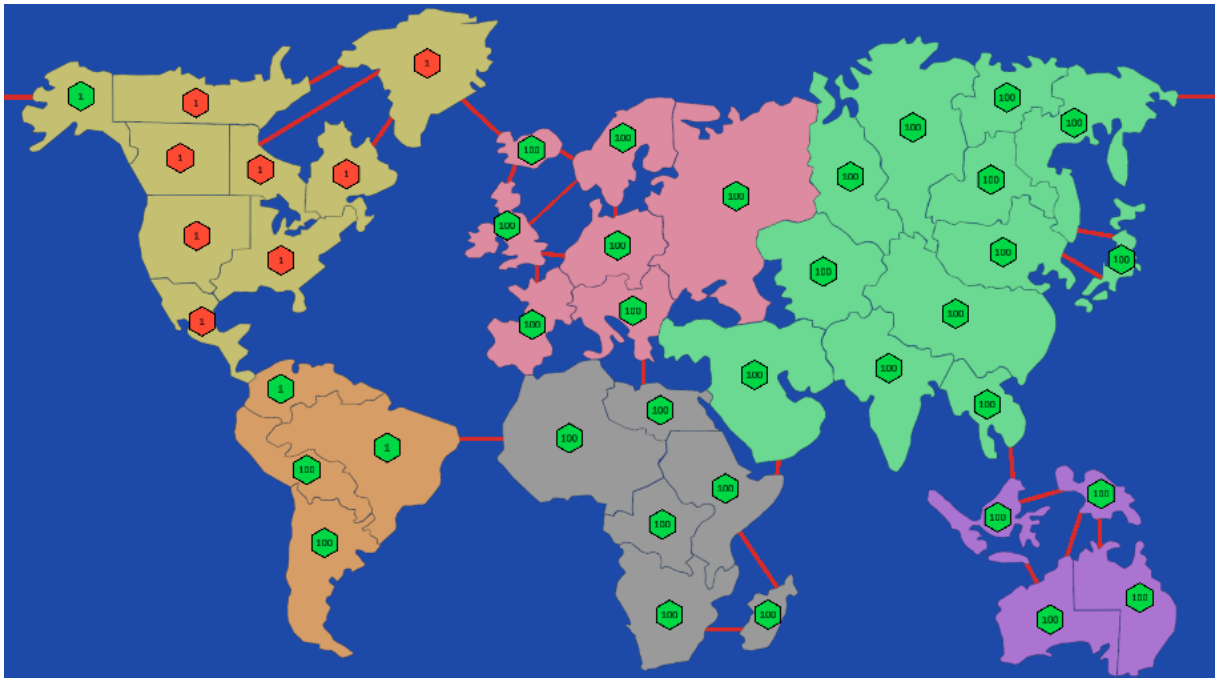


Fonte: Próprio autor.

Também existe um slider de controle da velocidade que os comando e turnos irão passar caso esteja ativo o modo de andamento automático. Pode ser visualizado na figura 5.5.

Para cada comando do jogador o mapa irá reagir de uma forma diferente, mostrando os resultados do comando ativo e realçando quais países estão participando. Também é mostrado na parte superior um texto com o comando executado. O comando de colocar novas tropas pode ser visualizado na figura 5.6, o comando de mover tropas pode ser visualizado na figura 5.7 e o comando de ataque pode ser visualizado na figura 5.8.

Figura 5.4 – Mapa mostrando o estado atual do jogo, países dominados e quantas tropas contém



Fonte: Próprio autor.

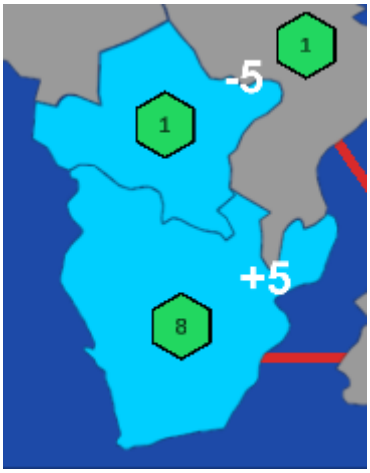
Figura 5.5 – Sliders de controle de velocidade do andamento automático

Fonte: Próprio autor.

Figura 5.6 – exemplo do mapa com o comando de colocar novas tropas executado

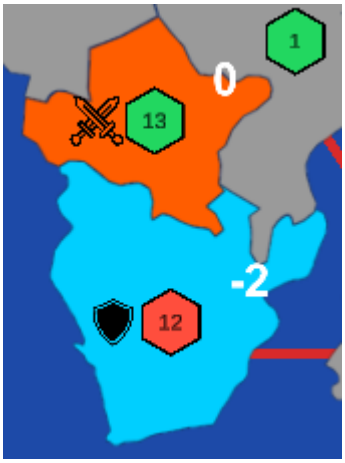
Fonte: Próprio autor.

Figura 5.7 – exemplo do mapa com o comando de mover tropas executado



Fonte: Próprio autor.

Figura 5.8 – exemplo do mapa com o comando de atacar tropas executado



Fonte: Próprio autor.

6 CONCLUSÃO

As competições de IA de jogos são motivadores importantes para pesquisa e desenvolvimento em IA de jogos e de IA em geral, pois lida com questões que podem ser aplicadas a problemas reais. As características do jogo Risk, permitem que competições de todos os níveis de dificuldade sejam aplicadas.

E é por estes motivos que neste trabalho de conclusão de curso foi desenvolvido um sistema capaz de proporcionar um ambiente completo para competições entre agentes, para o jogo de tabuleiro Risk. Este ambiente disponibiliza 3 componentes, jogo, visualizador gráfico e agentes de exemplo. Dessa forma estudantes e professores podem utilizar para desenvolver seus conhecimentos em IA.

Este sistema se diferencia de outros já conhecidos, pois implementa um jogo que tem poucos trabalhos e pesquisas sobre e também pela diferente comunicação entre suas partes (jogo e agentes), que é feita através de arquivos, o que permite uma maior liberdade na implementação de seus agentes, já que podem ser implementados em qualquer linguagem com suporte a arquivos.

Foram mostrados detalhes de como cada componente foi desenvolvido e ou executado, para que outros desenvolvedores possam melhorar cada um deles.

7 TRABALHOS FUTUROS

O sistema implementado neste trabalho, tem a possibilidade de ser expandido e melhorado posteriormente para ter mais desafios, confiabilidade e poder lidar com outros problemas de IA. A primeira possibilidade é sobre as funcionalidades do jogo, a versão do jogo utilizada neste trabalho, é uma versão simplificada, então existem funcionalidades que ainda podem ser introduzidas, como por exemplo, as cartas de território.

Elas funcionam da seguinte forma, cada país tem uma carta com um símbolo, existem 3 símbolos, símbolo de Artilharia, Cavalaria ou Infantaria. Além destas cartas existem mais 2 cartas coringas, mostrando todos os três símbolos ao mesmo tempo.

Estas cartas são ganhadas dominando países, apenas 1 por turno, caso o jogador consiga dominar algum país novo na etapa de ataque, ganhará uma carta aleatória. E quando o jogador juntar 3 cartas, ele poderá trocar por tropas novas na etapa de colocar tropas novas. A quantidade de tropas depende das combinações dos símbolos das cartas trocadas.

Tabela 7.1 – Quantidade de tropas bônus por cada possível troca (3 cartas)

Símbolo 1	**Símbolo 2**	**Símbolo 3**	**Tropas Bônus**
Infantaria	Infantaria	Infantaria	4
Cavalaria	Cavalaria	Cavalaria	6
Artilharia	Artilharia	Artilharia	8
Infantaria	Cavalaria	Artilharia	10

Fonte: Adaptado de (WOLF, 2005)

Outra possibilidade de expansão é facilitar a execução do jogo e dos agentes, eles são executados um de cada vez, agentes primeiros depois o jogo, uma alternativa seria implementar algum programa que faça essas 3 chamadas de uma única vez. Existe também a possibilidade de fazer o jogo decidir aleatoriamente qual jogador será o primeiro a jogar, diminuindo a desvantagem de quem joga em segundo lugar.

Em trabalhos futuros, poderiam ser feitos testes com o sistema em algum ambiente real, como em salas de aula ou eventos. Para assim poder ter um feedback completo do quanto o trabalho agregou para os estudantes e professores e no que poderia ser alterado para melhorar a experiência utilizando o sistema.

Um ponto possível de melhoria é sobre tornar o sistema mais seguro contra trapaças. No estado atual, o sistema já consegue garantir certa segurança, mas ainda existem melhorias a serem feitas neste quesito. Uma delas seria, conseguir monitorar qual agente acessou cada arquivo, para poder controlar concorrência de escrita em arquivos e também garantir que cada agente apenas modifique o seu respectivo arquivo.

Uma outra expansão que pode ser feita é na parte do visualizador e do jogo, atu-

almente eles apenas mostram a partida depois de finalizada, seria interessante criar um novo modo em que permita visualização da partida em tempo real.

Na parte do visualizador também poderia ser acrescentado alguns dados e estatísticas sobre a partida e os agentes, que fossem alterando durante todo o andamento da partida.

REFERÊNCIAS BIBLIOGRÁFICAS

AMRAI, K. et al. The relationship between academic motivation and academic achievement students. **Procedia - Social and Behavioral Sciences**, v. 15, p. 399–402, 2011. ISSN 1877-0428. 3rd World Conference on Educational Sciences - 2011. Disponível em: <https://www.sciencedirect.com/science/article/pii/S1877042811002904>.

BETTKER, R. et al. An implementation of the 7 wonders board game for ai-based players. In: . [S.l.: s.n.], 2020.

CARPIO, J. et al. Open classroom: Enhancing student achievement on artificial intelligence through an international online competition. **Journal of Computer Assisted Learning**, v. 31, 08 2014.

____. Open classroom: Enhancing student achievement on artificial intelligence through an international online competition. **Journal of Computer Assisted Learning**, v. 31, 08 2014.

CHESANI, F. et al. A game-based competition as instrument for teaching artificial intelligence. In: **AI*IA**. [S.l.: s.n.], 2017.

FACHADA, N. Colorshapelinks: A board game ai competition for educators and students. **Computers and Education: Artificial Intelligence**, v. 2, p. 100014, 2021. ISSN 2666-920X. Disponível em: <https://www.sciencedirect.com/science/article/pii/S2666920X21000084>.

FERRARI, R. G. **Uma comparação entre abordagens de IA para o jogo Risk**. 2022. Monografia (Trabalho de Graduação), 2022.

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

HASBRO. **Risk, The World Conquest Game**. [S.l.], 1993. 16 p. Acesso em 03 dez. 2021. Disponível em: <https://www.hasbro.com/common/instruct/risk.pdf>.

JUSTESEN, N. et al. Blood bowl: A new board game challenge and competition for ai. In: **2019 IEEE Conference on Games (CoG)**. [S.l.: s.n.], 2019. p. 1–8.

NEWBORN, M. Kasparov versus deep blue - computer chess comes of age. In: . [S.l.: s.n.], 1996.

O'RIORDAN, F.; KIRKLAND, D. Games as an engaging teaching and learning technique: Learning or playing. **Novel Approaches to Promoting Student Engagement**, Citeseer, v. 77, 2008.

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

REEVE, J. **Understanding motivation and emotion**. [S.l.]: Wiley, 2018.

RUSSELL, S. J. **Artificial intelligence: a modern approach**. [S.l.]: Upper Saddle River, NJ : Prentice Hall, 2010. 1132 p.

SALTA, A.; PRADA, R.; MELO, F. S. A game ai competition to foster collaborative ai research and development. **ArXiv**, abs/2010.08885, 2020.

____. A game ai competition to foster collaborative ai research and development. **IEEE Transactions on Games**, Institute of Electrical and Electronics Engineers (IEEE), v. 13, n. 4, p. 398–409, Dec 2021. ISSN 2475-1510. Disponível em: <http://dx.doi.org/10.1109/TG.2020.3024160>.

Sillysoft. **Lux Delux**. Sillysoft, 2002. Acessado em 03 dez 2021. Disponível em: <https://sillysoft.net/lux/>.

SILVER, D. et al. Mastering the game of go with deep neural networks and tree search. **Nature**, v. 529, p. 484–489, 01 2016.

TOGELIUS, G. N. Y. J. **Artificial Intelligence and Games**. [S.l.]: Springer, 2018. 337 p.

TURING, A. M. I.—COMPUTING MACHINERY AND INTELLIGENCE. **Mind**, LIX, n. 236, p. 433–460, 10 1950. ISSN 0026-4423. Disponível em: <https://doi.org/10.1093/mind/LIX.236.433>.

WOLF, M. M. An intelligent artificial player for the game of risk. In: . [S.l.: s.n.], 2005.

ŚWIECHOWSKI, M. Game ai competitions: Motivation for the imitation game-playing competition. In: . [S.l.: s.n.], 2020.