Relatório sobre IAs para o jogo WAR (pesquisa inicial)

1) Contextualização

1.1) Domínio do jogo WAR

Primeiramente temos que estabelecer o domínio do problema, garantindo que a IA seja projetada com um profundo entendimento da mecânica e das nuances estratégicas do jogo WAR. Para isso, vamos definir o conjunto de informações que são relevantes para a IA tomar decisões durante a partida:

- **Territórios e Conectividade:** O tabuleiro do jogo pode ser modelado como um grafo não direcionado, onde os territórios são os nós e as adjacências são as arestas. Essa representação de grafo é crucial para algoritmos de busca de caminho, análise de fronteiras e serve como uma entrada natural para arquiteturas de IA mais avançadas, como Redes Neurais de Grafos (Graph Neural Networks - GNNs).

- **Exércitos e Propriedade:** A distribuição de exércitos pelo grafo e a posse de cada nó por um jogador são os elementos dinâmicos centrais. A quantidade de exércitos em um território determina sua força ofensiva e defensiva.

- **Continentes:** Definidos como subgrafos específicos no tabuleiro. O controle completo de um continente confere uma vantagem significativa de recursos (exércitos de bônus), tornando-os objetivos estratégicos chave. A estrutura do grafo determina a defensibilidade de cada continente, com aqueles que possuem menos pontos de entrada (arestas conectando a nós fora do subgrafo), como a Austrália, sendo os mais cobiçados.

- **Cartas:** Um sistema de recursos secundário que introduz tanto aleatoriedade (a carta comprada após uma conquista) quanto profundidade estratégica (quando trocar conjuntos por exércitos). O valor crescente dos conjuntos de cartas é um fator crítico no ritmo do jogo, podendo levar a viradas dramáticas.

1.2) Pontos de decisão

Além disso, precisamos levar em conta os pontos de decisão da IA, ou seja, as etapas do jogo em que ela escolherá uma ação para tomar. Cada fase do turno de um jogador apresenta um subproblema distinto para a IA resolver. Uma IA sofisticada não deve tratar essas fases como problemas de otimização isolados. As decisões tomadas em uma fase impactam diretamente as opções e a eficácia nas fases subsequentes, criando uma cadeia causal: o reforço prepara um ataque, o ataque cria uma nova fronteira para fortificar, e a fortificação posiciona as forças para o reforço do próximo turno. Portanto, a avaliação de uma sequência de movimentos deve considerar o estado do tabuleiro ao final do turno, e não apenas após uma única ação.

- **Fase de Reforço:** Essencialmente, um problema de alocação de recursos. A IA deve decidir onde posicionar novos exércitos para maximizar a segurança defensiva e o potencial ofensivo. Esta decisão deve antecipar os ataques planejados e as potenciais ameaças do oponente.
- **Fase de Ataque:** Um problema de tomada de decisão sequencial sob incerteza. A IA deve decidir *se* deve atacar, *qual* território inimigo adjacente atacar, *de onde* e *quantas vezes*. O resultado de cada ataque é estocástico, determinado por rolagens de dados, o que exige um raciocínio probabilístico.
- **Fase de Fortificação:** Um problema de reposicionamento estratégico. A IA pode fazer um único movimento para consolidar forças, fortalecer uma fronteira vulnerável ou preparar uma ofensiva para o próximo turno.

1.3) Espaço de problemas para a IA

Com esses pontos estabelecidos, podemos analisar qual é o espaço de problemas que a IA terá que lidar no jogo WAR. Caracterizar o WAR usando a terminologia formal de IA ajuda a enquadrar as discussões técnicas subsequentes e a selecionar as abordagens apropriadas.

- **Alto Fator de Ramificação:** Em qualquer ponto do jogo, um jogador pode ter dezenas ou centenas de combinações de movimentos possíveis (alocar N exércitos em M territórios, escolher entre K ataques, etc.). Isso torna os métodos de busca exaustiva, como o Minimax básico, computacionalmente inviáveis.

- **Elementos Estocásticos:** As rolagens de dados introduzem aleatoriedade, o que significa que uma ação específica não tem um resultado determinístico. Isso invalida algoritmos de busca determinísticos simples e exige raciocínio probabilístico.

- **Ambiente Multiagente e de Soma Não-Zero:** Diferente de jogos de dois jogadores e soma zero como o xadrez, o WAR normalmente envolve de 3 a 6 jogadores. Uma ação que prejudica um oponente pode beneficiar outro inadvertidamente. Isso complica a avaliação, pois a IA não pode simplesmente assumir que todos os outros jogadores agirão para minimizar sua pontuação.

- **Informação Imperfeita (em algumas variantes):** Embora o estado do tabuleiro seja geralmente totalmente observável, o conteúdo das cartas nas mãos dos outros jogadores está oculto, adicionando uma camada de informação imperfeita que requer inferência e modelagem do oponente.

2) Alguns tipos de IAs

2.1) Agentes Baseados em Heurísticas

Vamos começar falando sobre os modelos baseados em heurísticas, que são muito úteis para criar IAs simples, que podem ser usadas como IAs básicas para benchmarks iniciais com uma IA de dificuldade “fácil”.

A busca heurística é um conceito central da IA que utiliza "regras práticas" para guiar a tomada de decisão em grandes espaços de busca, evitando o custo computacional dos métodos de força bruta. Para o WAR, essas heurísticas são derivadas de estratégias humanas estabelecidas.

O "cérebro" de uma IA heurística é sua função de avaliação, que atribui uma pontuação numérica a um determinado estado do jogo da perspectiva da IA. Uma pontuação mais alta indica uma posição mais favorável. As decisões são tomadas escolhendo a ação que leva ao estado com a maior pontuação.

2.1.1) Prós e Contras da IA Heurística

- **Prós:** Simples de implementar, computacionalmente barata, previsível e altamente depurável, excelente para níveis de dificuldade base e fácil/médio.
- **Contras:** Pode ser previsível e explorável por jogadores experientes, requer conhecimento de domínio significativo para elaborar as regras, é frágil (não consegue se adaptar a estratégias novas) e é difícil equilibrar os pesos perfeitamente.

2.2) Minimax e Expectiminimax

O algoritmo Minimax é uma busca recursiva e em profundidade da árvore de jogo, onde os nós MAX (IA) e MIN (oponente) se alternam. A IA assume que o oponente sempre fará o movimento ideal para minimizar a pontuação da IA. A Poda Alfa-Beta é uma otimização crítica que "poda" ramos da árvore de busca que não podem influenciar a decisão final, tornando buscas mais profundas viáveis.

Apesar de sua importância teórica, o Minimax clássico falha em WAR por duas razões fundamentais:

- **N-Jogadores, Soma Não-Zero:** O Minimax assume um jogo de dois jogadores e soma zero. Em WAR, com múltiplos oponentes, não é possível ter um único jogador "MIN". Uma ação que é ruim para o oponente A pode ser boa para o oponente B, quebrando a premissa central do algoritmo.

- **Resultados Estocásticos:** O Minimax assume movimentos determinísticos. Em WAR, um ataque não é um único resultado, mas uma distribuição de probabilidade de resultados. O Minimax não pode lidar com isso nativamente.

A combinação do alto fator de ramificação com essas limitações teóricas revela uma lacuna de praticidade. Uma possível alternativa seria usar o Expectiminimax, para lidar com a parte estocástica do jogo, já que esse algoritmo introduz um terceiro tipo de nó: um nó de "Acaso". Em vez de pegar o mínimo ou o máximo dos valores de seus filhos, um nó de acaso calcula o *valor esperado* (uma média ponderada com base na probabilidade de cada resultado).

2.2.1) Prós e Contras do Expectiminimax

- **Prós:** Fornece uma maneira matematicamente sólida de raciocinar sobre eventos aleatórios, sendo mais robusto do que uma heurística simples em situações táticas.
- **Contras:** Sofre do mesmo fator de ramificação massivo do Minimax, tornando buscas profundas computacionalmente proibitivas. O problema de n-jogadores permanece. A poda alfa-beta é menos eficaz porque o valor esperado de um nó de acaso pode estar dentro dos limites alfa-beta, mesmo que alguns resultados individuais de alta probabilidade estejam fora deles.

2.3) Busca em Árvore Monte Carlo (MCTS)

Esse algoritmo é um dos mais promissores para se implementar uma IA escalável e de alto desempenho para o jogo de WAR. MCTS é um algoritmo de busca heurística que aborda o problema do alto fator de ramificação não explorando toda a árvore de jogo. Em vez disso, ele usa amostragem aleatória (simulações) para construir uma árvore de busca assimétrica, concentrando seu esforço computacional nas linhas de jogo mais promissoras.

Sua principal vantagem é ser um algoritmo "anytime". Ele pode ser interrompido a qualquer momento para retornar o melhor movimento atual, e a qualidade desse movimento geralmente melhora com mais tempo de simulação. Isso o torna perfeito para níveis de dificuldade personalizáveis: uma IA "Fácil" tem 100ms para pensar, enquanto uma IA "Difícil" tem 2 segundos.

2.3.1) Prós e Contras do MCTS

- **Prós:** Desempenho de ponta em jogos complexos, lida bem com altos fatores de ramificação e estocasticidade, a propriedade "anytime" é ideal para escalar a dificuldade, não requer uma função de avaliação complexa e artesanal (embora se beneficie de uma nos rollouts).
- **Contras:** Pode ser computacionalmente caro, o desempenho depende do número de simulações, pode ocasionalmente perder erros táticos de curto prazo que uma busca de largura total como o Minimax detectaria (um fenômeno conhecido como "armadilhas rasas") e pode ser mais complexo de implementar e depurar do que um agente heurístico simples.

2.4) Agentes evolutivos e de aprendizagem

Esses tipos de agente são construídos a partir de técnicas onde a estratégia da IA não é explicitamente programada, mas é aprendida ou evoluída através de processos automatizados. Estes métodos são adequados para criar oponentes de alto nível e imprevisíveis.

2.4.1) Algoritmos Genéticos (GA):

Os Algoritmos Genéticos são inspirados na seleção natural. Uma "população" de agentes de IA, cada um com um conjunto diferente de pesos para sua função de avaliação, é criada. Esses agentes jogam uns contra os outros em um torneio. O ciclo evolutivo consiste em:

1. **Avaliação (Função de Aptidão):** A "aptidão" de cada agente é seu desempenho no torneio (por exemplo, taxa de vitórias).
2. **Seleção:** Os agentes com melhor desempenho são selecionados para "reproduzir".
3. **Crossover:** Novos agentes ("descendentes") são criados combinando os pesos da função de avaliação de dois agentes "pais".
4. **Mutação:** Pequenas alterações aleatórias são introduzidas nos pesos dos descendentes para garantir a diversidade genética.

Este processo é repetido por muitas gerações, evoluindo gradualmente um conjunto altamente otimizado de pesos para a função de avaliação heurística, potencialmente descobrindo estratégias que um humano poderia não perceber.

2.4.2) Aprendizado por Reforço (RL):

O Aprendizado por Reforço é um paradigma onde um agente aprende a tomar decisões ótimas interagindo com um ambiente e recebendo recompensas ou punições por suas ações. Ele aprende uma "política" que mapeia estados de jogo para ações.

No processo de treinamento, o agente de IA joga milhões de jogos contra si mesmo ("auto-jogo"). Inicialmente, seus movimentos são aleatórios. Com o tempo, ele aprende quais sequências de ações levam a recompensas cumulativas mais altas e ajusta sua política interna (muitas vezes representada por uma rede neural) de acordo.

2.4.3) Prós e Contras de Agentes de aprendizagem/Evolutivos

- **Prós:** Podem alcançar desempenho sobre-humano, descobrir estratégias novas e altamente eficazes e são menos dependentes do conhecimento de domínio humano (especialmente RL).
- **Contras:** Extremamente caros computacionalmente para treinar (requerem a simulação de milhões de jogos), podem ser uma "caixa preta", tornando difícil depurar ou entender sua estratégia, e exigem um ambiente de simulação robusto separado do cliente do jogo.

2.5) Abordagens híbridas (usando combinações de técnicas):

Um outro modo de se projetar IAs para jogos, é combinando o melhor de cada algoritmo, criando assim uma solução mais completa. Por exemplo, uma função de avaliação heurística melhora os rollouts do MCTS ; o Minimax melhora o jogo tático do MCTS ; e uma rede neural pode substituir tanto a função de avaliação quanto a política de rollout no MCTS, tornando-o ordens de magnitude mais forte.

2.5.1) Prós e Contras de abordagens híbridas:

- **Prós:** Como são técnicas de ponta que combinam os pontos fortes de diferentes modelos de IA, pode ser alcançado um desempenho maior que a soma de suas partes.
- **Contras:** Alto nível de complexidade.

3) O Fator Humano: Criando um Oponente Cativante

No projeto de uma boa IA, também temos que levar em conta pontos que vão além do desempenho puro e discutir a arte de projetar uma IA que seja agradável e desafiadora, não apenas impiedosamente eficiente.

Sendo assim, é importante que os movimentos da IA sejam compreensíveis pelos jogadores humanos, permitindo a elaboração de contra-jogadas e interações estratégicas entre humanos e IAs. Dessa forma, uma possível abordagem, é criar IAs com determinadas personalidades, como por exemplo uma IA "Agressiva" pode se comprometer demais com ataques e deixar suas fronteiras fracas. Uma IA "Cautelosa" pode fortificar demais e perder oportunidades de expansão. Porém, essas “falhas” devem ser consistentes com a persona da IA.

4) Implementando uma IA em um projeto usando Java Spring Boot

Como trabalhamos com a arquitetura Cliente-Servidor quando usamos o framework Spring Boot, teremos o sistema dividido basicamente em duas partes:

Frontend (Navegador): Será o "cliente". Sua responsabilidade é puramente a apresentação (renderização) do tabuleiro e a captura de inputs do jogador humano. Ele não terá a lógica do jogo ou da IA. Ele se comunicará com o backend através de uma API.

Backend (Java/Spring Boot): Será o "servidor" e o cérebro do jogo. Suas responsabilidades são:

- Manter o estado oficial e completo do jogo (quem possui qual território, quantas tropas, etc.).
- Validar as jogadas enviadas pelos jogadores.
- Executar as regras do jogo.
- Hospedar e executar os cálculos para os jogadores de IA.
- Gerenciar a persistência do jogo (salvar e carregar partidas).

4.1) Como Implementar a IA no Backend Spring Boot

4.1.1) Comunicação em Tempo Real com WebSockets

Para um jogo de tabuleiro, uma API REST tradicional pode ser lenta. A melhor abordagem é usar **WebSockets**, que permitem uma comunicação bidirecional e em tempo real. O Spring Boot tem um excelente suporte para isso.

- **Fluxo de Jogo:**

1. O jogador humano faz uma jogada no frontend.
2. O frontend envia a ação para o backend via WebSocket (ex: `{"action": "ATTACK", "from": "BRAZIL", "to": "ARGENTINA"}`).
3. O servidor (Spring Boot) recebe a ação, valida e atualiza o estado do jogo.
4. O servidor então transmite o novo estado do jogo para todos os clientes conectados àquela partida via WebSocket.
5. Quando for a vez de um jogador IA, o servidor executa a lógica da IA, atualiza o estado e novamente transmite o novo estado para todos.

4.1.2) O Padrão de Projeto "Strategy" para a IA

Esta é a forma mais limpa e escalável de organizar sua IA em Java. O **Strategy Design Pattern** permite que você

encapsule diferentes algoritmos (nossos níveis de dificuldade da IA) em objetos separados. Além disso, podemos usar a inversão de dependência injetando e usando a estratégia apropriada para cada jogador IA.

4.1.3) Multi-threading em Java

O cálculo de uma IA complexa (como MCTS) pode levar alguns segundos, e para que isso não provoque que o servidor pare de responder a outras requisições nesse meio tempo, podemos usar multi-threading em Java.

5) Artigos relacionados ao assunto:

Design Evolutivo de Jogos de Mesa: Um Estudo de Caso no Jogo Risk:
https://lume.ufrgs.br/bitstream/handle/10183/263286/001175020.pdf?sequence=1

OBS: Risk é o jogo que serviu de inspiração para a criação do jogo WAR, o que faz com que eles sejam bem semelhantes.

UM FRAMEWORK PARA PESQUISA DE INTELIGÊNCIA ARTIFICIAL EM JOGOS DE ESTRATÉGIA POR TURNOS:
https://d1wqtxts1xzle7.cloudfront.net/53678764/1432M-libre.PDF?1498576118=&response-content-disposition=inline%3B+filename%3DUM_FRAMEWORK_PARA_PESQUISA_DE_INTELIGENC.pdf&Expires=1757291769&Signature=WrT~hwYA1EnZZmJrPZkYG1BQo6sWhYf92xG9jc3pIEb8yWat-Jb7mgvNl15Rnwme-hFsOO~-CEGuvje7i76xPAoBFmxfI8towolFo2TcIDM89LEr8N7HrFw3KF~K6Y-So5fYaPvHMz1JDUCy1nc~gaiT-KF9bC5J2yvwaFW6e-99MM4s2yGTchUmq099LfggI9FA2HgN2ia48DDZdPjJMtTmSI8oksDBk757~9q3iGWluSq8PFmmWA2nTfPa0a2slv9dVc0RdBN6279Ydr2O1Dgxbn6XUjy0HQzyTyOietG3nrijRlw9KQkpOcB-dwL58nTKNipOMP4qEZoQ0EWIlw__&Key-Pair-Id=APKAJLOHF5GGSLRBV4ZA

Essa dissertação fala sobre um framework para ser usado para o desenvolvimento de IAs para jogos de estratégia baseados em turnos.

UM SISTEMA DE APOIO AO JOGADOR PARA JOGOS DE ESTRATÉGIA EM TEMPO REAL:
https://repositorio.ufmg.br/bitstream/1843/SLSS-85RK8D/1/renatoluizdefreitascunha.pdf

Essa dissertação fala sobre IAs para ajudar os jogadores, o que pode ser um extra interessante para colocar no projeto, caso sobre tempo para isso.

INTELIGÊNCIA ARTIFICIAL ADAPTATIVA PARA AJUSTE DINÂMICO DE DIFICULDADE EM JOGOS DIGITAIS:
https://repositorio.ufmg.br/bitstream/1843/ESBF-AARQWM/1/mirnapaula.pdf

Essa dissertação fala sobre uma abordagem útil para ajustar dinamicamente o nível de dificuldade da IA conforme o desempenho dos jogadores durante a partida, para oferecer uma experiência desafiadora, mas ao mesmo tempo divertida e de acordo com o nível de habilidade do jogador.

Uma Aplicação de Inteligência Artificial Minimax para Jogos de Tabuleiro Abstratos e de Estratégia:
https://lume.ufrgs.br/bitstream/handle/10183/218630/001123316.pdf?sequence=1

Essa monografia fala sobre a aplicação do algoritmo de IA Minimax em jogos de tabuleiros.