Sampling battleships boards for hit estimation using Monte Carlo Tree Search

Eduardo Naslausky[1], **Daniel Ratton Figueiredo**[1]

[1]Programa de Engenharia de Sistemas e Computação (PESC)
Universidade Federal do Rio de Janeiro (UFRJ)
Rio de Janeiro – RJ

{naslausky, daniel}@cos.ufrj.br

Abstract. *Battleships is a classic board game for two people. One of the main problems of the game is to determine the next position on the board to take your turn and be revealed by the opponent. Intuitively, the chosen position should contain a boat (hit), seen as the objective of the game is to sink all enemy boats. This work uses Monte Carlo Tree Search algorithm to sample valid boards with all boats from a given initial empty board (current game state). The hit estimation for every hidden board cell is made by using the sampled boards. To evaluate the quality of this algorithm, matches have been simulated where every turn uses the best estimated hit coordinate from the previous turn. Results indicate a 65% acuracy for the top-1 coordinate after one turn and it grows monotonically, reaching 80% after 10 turns.*

Resumo. *Batalha Naval é um jogo clássico de tabuleiro com duas pessoas. Um dos problemas fundamentais do jogo é determinar a próxima posição do tabuleiro a ser revelada pelo adversário. Intuitivamente, a posição escolhida deveria corresponder a um barco (acerto), tendo em vista o objetivo do jogo de afundar todos os barcos do adversário. Este trabalho utiliza o algoritmo de Monte Carlo Tree Search para gerar amostras de tabuleiros válidos com todos os barcos a partir de um tabuleiro inicial (configuração atual do jogo). A probabilidade de acerto de cada posição oculta do tabuleiro é calculada utilizando as amostras geradas. Para avaliar a qualidade deste algoritmo, partidas foram simuladas onde cada jogada utiliza a posição de maior probabilidade de acerto do turno anterior. Resultados indicam que a acurácia (top-1) depois de uma jogada é cerca de 65% e cresce monotonicamente, chegando a 80% depois de 10 jogadas.*

1. Introdução

Batalha Naval (*Battleships*, em inglês) é um jogo clássico para dois jogadores, comercializado há quase um século [1]. Cada jogador possui um tabuleiro quadriculado e um conjunto de barcos que devem ser colocados no tabuleiro de forma que cada posição do tabuleiro é ocupada exatamente por um barco ou por água. Uma vez posicionado os barcos de ambos jogadores, que permanecem imóveis durante a partida, os jogadores então alternam entre si informando coordenadas do tabuleiro em que acreditam que os barcos inimigos possam estar. O jogador que recebe o disparo confere em seu tabuleiro e informa ao outro se foi

[1]Ver página do jogo na Wikipedia.

um acerto (dano a algum barco) ou um erro (água). Um barco é considerado afundado quando todas suas respectivas posições no tabuleiro foram atingidas. Vence o jogador que afundar todos os barcos do outro primeiro. Jogos mais modernos de batalha naval contam com opções de habilidades e diferentes formatos de barcos e tabuleiros, mas a essência do jogo continua a mesma. Apesar de ser um jogo, o mesmo vem sendo abordado na literatura acadêmica por diferentes motivos (ver discussão na seção 4).

Um problema fundamental deste jogo é determinar qual posição do tabuleiro um jogador deve jogar [Crombez et al. 2020]. Intuitivamente, a posição a ser escolhida deve ser aquela com a maior chance de ter um barco do oponente. Ou seja, dada uma configuração do tabuleiro do oponente, com posições marcadas por acertos e erros, precisamos determinar a chance de ter barco em cada uma das posições ocultas (ainda não jogadas). Esta chance pode ser calculada considerando todos os possíveis arranjos de barcos do oponente que são compatíveis com a configuração atual do tabuleiro. Uma forma de enumerar os arranjos é construindo uma árvore onde o nó raiz é o tabuleiro atual do oponente, e os nós folhas são tabuleiros válidos com todos os barcos do oponente posicionados. Ao colocar um barco de forma consistente com um tabuleiro, criamos um nó filho. Dessa forma, temos uma árvore que constrói todos os possíveis tabuleiros consistentes com a raiz. Repare que a altura da árvore é dada pelo número de barcos do oponente que ainda não foram afundados.

Entretanto, o número de nós desta árvore em geral é muito grande, principalmente quando a configuração do tabuleiro raiz possui poucas posições marcadas por acertos e erros. Neste caso, o número de filhos do nó raiz é muito alto (centenas), já que existem muitos possíveis arranjos consistentes para adicionar um dos barcos do oponente. Dessa forma, calcular o número de folhas dessa árvore se torna computacionalmente inviável [Sevenster 2004].

Um outro aspecto importante no cálculo da chance de uma posição do tabuleiro ter um barco é a chance do oponente escolher uma determinada configuração de tabuleiro. Intuitivamente, as possíveis configurações de tabuleiro não são equiprováveis, pois um jogador muito provavelmente prefere espalhar seus barcos pelo tabuleiro do que colocar todos os barcos lado-a-lado. Dessa forma, as possíveis configurações de tabuleiro (folhas da árvore acima) devem ser ponderadas para representar a chance do oponente escolher tal configuração.

O algoritmo de *Monte Carlo Tree Search* (MCTS) [Browne et al. 2012, Świechowski et al. 2023] é frequentemente utilizado para fazer buscas e estimativas em árvores de jogo que são enormes. Em sua forma mais comum, o MCTS é utilizado para estimar a chance de vencer a partida se o jogador tomar uma certa ação dado a configuração atual do tabuleiro (nó da árvore do jogo). Neste caso, o MCTS usa aleatoriedade para simular um grande número de partidas independentes a partir do tabuleiro atual até o final do jogo (folhas da árvore), e com isso estimar a chance de vencer a partida para cada possível ação do tabuleiro atual.

No caso de batalha naval, estamos interessados em estimar a chance de haver um barco em cada uma das posições ocultas do tabuleiro atual. Entretanto, podemos utilizar o algoritmo MCTS para gerar um grande número de amostras independentes de tabuleiro consistentes com o atual. Além disso, essa geração de amostra é aleatória e enviesada

(não é equiprovável), pois deve capturar formas mais prováveis do oponente colocar seus barcos no tabuleiro. Utilizando as amostras geradas (tabuleiros consistentes), podemos calcular a fração de tabuleiros que possui um barco em cada uma das posições ocultas do tabuleiro atual. Além disso, podemos calcular o número total de tabuleiros consistentes distintos que foram gerados.

O objetivo deste trabalho é adaptar o algoritmo MCTS para gerar amostras enviesadas de tabuleiros consistentes para calcular as posições ocultas mais prováveis de terem barco na configuração atual. Propomos uma metodologia de como realizar esta amostragem de forma enviesada, utilizando diferentes heurísticas que se baseiam no conhecimento do jogo. A metodologia é avaliada numericamente em diferentes cenários (tabuleiro atual), reportando a acurácia nas posições mais prováveis (top-1, top-3 e top-5). Os resultados são promissores e indicam que a metodologia proposta consegue encontrar coordenadas contendo barcos em boa parte dos casos.

O restante deste artigo está organizado da seguinte maneira. A seção 2 apresenta detalhes da versão do jogo usada neste trabalho e a metodologia para geração das amostras. A seção 3 apresenta a metodologia de avaliação assim como os resultados encontrados em diferentes cenários. Uma breve discussão dos trabalhos relacionados é apresentada na seção 4. Por fim, a seção 5 apresenta uma breve conclusão.

2. Jogo e Metodologia

Esta seção apresenta detalhes do jogo específico de Batalha Naval que será considerado neste trabalho, assim como a metodologia empregada para estimar a chance de acerto para cada posição oculta do jogo, dado a configuração atual do tabuleiro.

2.1. Formato do tabuleiro

Uma das diferenças que as variações dos jogos de Batalha Naval possuem é o tamanho do tabuleiro, que determina o número de posições (coordenadas). Por exemplo, um tabuleiro quadriculado de 10x10 (dez por dez) possui 100 posições (coordenadas). O tamanho do tabuleiro varia bastante entre versões do jogo, e algumas variações possuem tabuleiros que não são quadrados. Neste trabalho, o tamanho escolhido foi de 8x8, totalizando 64 posições distintas.

Além disso, algumas versões do jogo implementam a funcionalidade de pedras no tabuleiro. Esta funcionalidade marca algumas posições como ocupadas por uma pedra, impossibilitando um jogador de posicionar algum barco nelas. Essas coordenadas são preestabelecidas e podem ser alternadas a cada partida, mas são de conhecimento do jogador oponente. Um conjunto específico de pedras no tabuleiro é chamado de ”mapa”, e um exemplo de mapa é apresentado na figura 1.

2.2. Formatos e quantidade dos barcos

Outra variação comum é o formato e a quantidade de barcos que cada jogador dispõe para colocar no mapa. Os jogos mais clássicos contam apenas com alguns barcos em linha (i.e. que ocupam apenas uma coordenada de largura) de comprimento variado, com nomes muitas vezes dados em homenagem a embarcações reais (e.g. porta-aviões, cruzador, submarino, etc). Outras versões porém, contam com formatos mais complexos e distintos. Este trabalho não faz restrição quanto ao formato dos barcos do jogo, podendo estes serem

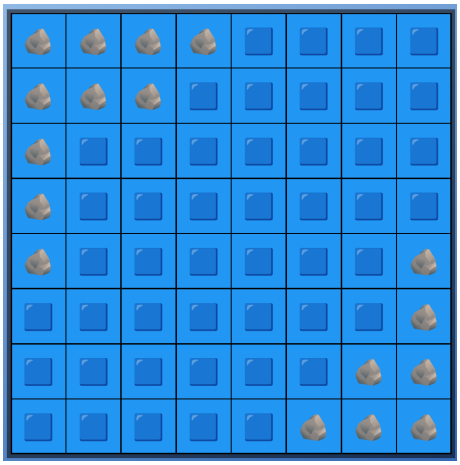

Figura 1. Exemplo de um mapa de um tabuleiro com pedras.

mais complexos. A figura 2 apresenta os formatos de barco disponíveis consideradas neste trabalho. A quantidade de barcos foi fixada em 4.

Figura 2. Formato dos barcos disponíveis. Jogador escolhe quatro barcos.

2.3. Habilidades

Uma variação muitas vezes encontrada é dar aos barcos diferentes habilidades. Uma dessas habilidades é o ”sonar”, que permite ao jogador obter conhecimento sobre o tabuleiro da seguinte forma. Em seu turno, caso o jogador tenha à sua disposição o sonar (e.g. tendo um barco não afundado com esta habilidade), ele pode utilizá-lo. A seguir, ele seleciona uma coordenada alvo para disparar o sonar. O oponente então revela se esta coordenada possui um barco ou não, da mesma forma que ocorreria com um tiro simples. Em adição, caso esta coordenada não possua um barco, é informada a menor distância desta coordenada para outra que contenha um barco. A distância informada é a distância de Manhattan [Moser 1982]. Isto diz que obrigatoriamente existe algum barco com uma distância específica da coordenada em que foi disparado o sonar.

A figura 3 apresenta o efeito de um sonar que foi disparado no canto inferior esquerdo do tabuleiro, e como retorno foi informada uma distância de duas unidades. Isto significa que as coordenadas marcadas como água obrigatoriamente não possuem

nenhum barco (por estarem a uma unidade de distância), e ao mesmo tempo alguma das coordenadas em amarelo contém algum barco.

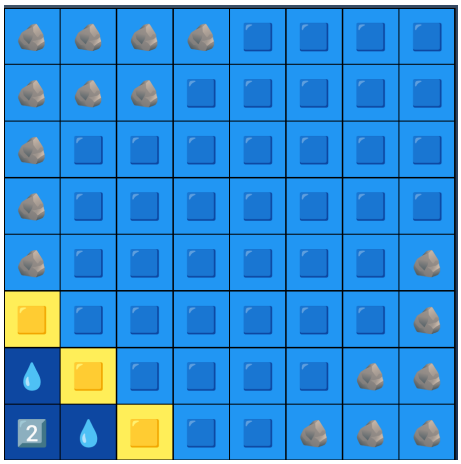

Figura 3. Exemplo de sonar que resultou em um valor de distância 2.

2.4. Aplicativo do jogo

Um aplicativo de celular do jogo de Batalha Naval com as variações descritas acima foi previamente desenvolvido por um dos autores. O aplicativo possui um módulo para calcular as chances de haver um barco em cada posição oculta do tabuleiro. Para fazer este cálculo, o aplicativo recebe a configuração do tabuleiro atual, as restrições de posicionamento de barcos relacionadas ao jogo, e os formatos dos barcos do oponente. O resultado é apresentado como um mapa de calor pelo aplicativo, indicando as posições de maior chance de terem um barco, assim como o total de tabuleiros consistentes que foram encontrados a partir do tabuleiro atual. A figura 4 ilustra a tela do aplicativo com as chances das posições ocultas.

2.5. Algoritmo MCTS

Uma variação do algoritmo MCTS é implementada neste trabalho, para gerar amostras de tabuleiros consistentes. A partir das amostras geradas, é calculado a chance de cada posição oculta do tabuleiro possuir um barco. No que segue, descrevemos seu funcionamento. O código do algoritmo está disponível publicamente no Github do autor.

O algoritmo 1 apresenta a variação do Monte Carlo Tree Search adotada neste trabalho. Primeiramente, um dicionário com todas as posições possíveis de cada barco no estado atual do tabuleiro é criado. Então, durante 10 segundos, o algoritmo explora a árvore de possibilidades construindo tabuleiros consistentes de maneira iterativa. O valor de 10 segundos foi escolhido de forma arbitrária, para que a estimativa pudesse ser realizada sem muita demora. Entretanto, este é um parâmetro do algoritmo.

Para explorar a árvore de tabuleiros, um barco é adicionado de cada vez ao tabuleiro atual (nó da árvore), escolhendo aleatoriamente sua posição dentre as posições possíveis que constam no dicionário. Entretanto, esta escolha aleatória não é uniforme, e sim feita de forma ponderada dependendo da configuração do tabuleiro com o barco a ser adicionado. Uma função de pontuação da configuração determina o peso da configuração, e a chance desta configuração ser escolhida é proporcional ao peso. A

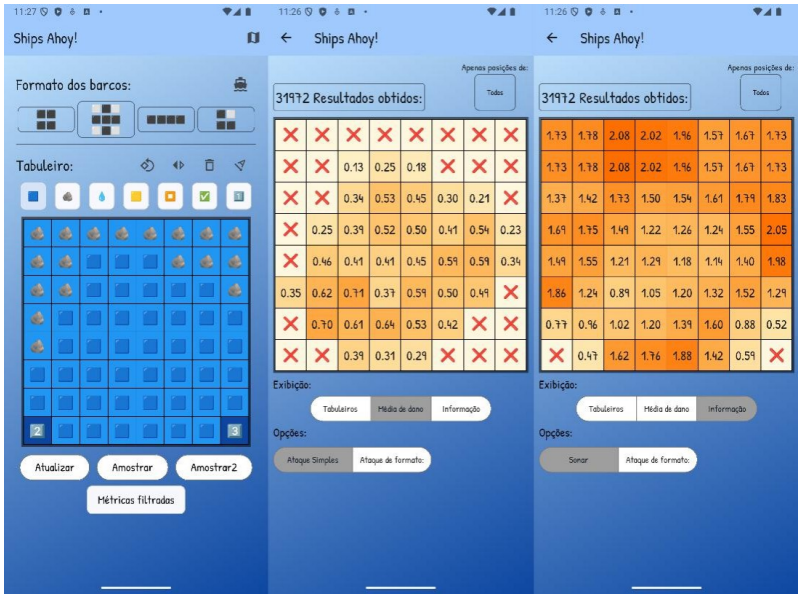

Figura 4. Imagens do aplicativo com a tela para definir a configuração inicial do tabuleiro (esquerda) e o mapa de calor com o resultado do cálculo da chance de ter barco.

função de ponderação será detalhada a seguir. Após a adição de todos os barcos, caso o tabuleiro seja válido (consistente), este é armazenado na lista de retorno. Repare que este tabuleiro corresponde a uma folha da árvore de tabuleiros. Detalhes da implementação estão no algoritmo 1.

2.6. Função de pontuação de um tabuleiro

O algoritmo MCTS utiliza uma função de pontuação para cada tabuleiro. Essa função é responsável por dar maior chance a tabuleiros intermediários que sejam mais promissores na obtenção de um tabuleiro válido (consistente) ao final da iteração. Esta função recebe um tabuleiro em seu estado atual (podendo ter alguns barcos já colocados), e a posição do novo barco a ser adicionado ao tabuleiro. A função retorna o peso do tabuleiro com o novo barco. Pesos maiores tem maior chance de serem selecionados pelo algoritmo MCTS (linearmente).

A função utilizada calcula o número de restrições sendo atendidas e o valor de um tabuleiro depende do tipo e do número de restrições atendidas. Foram utilizados dois tipos de restrições.

O primeiro tipo é o número de posições danificadas atendidas com o novo barco. Isto é, uma coordenada recebeu um tiro e o oponente informou que existe um barco nesta posição. Se a posição para o novo barco contém uma dessas posições danificadas, sua pontuação é aumentada. Vale deixar claro que este incremento é proporcional a quantidade de posições indicadas como barco que são atendidas. Ou seja, uma posição candidata que coincida com três posições que tem barco terá o triplo de pontuação de uma posição candidata que coincida com apenas uma posição.

O segundo tipo de restrição é relacionado a mecânica de “sonar”. Se a posição para o novo barco atende a distância dos sonares, então sua pontuação é incrementada de

Algorithm 1 Algoritmo de Monte Carlo

$barcos$: Lista dos barcos.
$inicial$: Estado inicial do tabuleiro (sem barcos).
$posBarcos \leftarrow \{\}$ ▷ Dicionário com as posições possíveis de cada barco.
$resultado \leftarrow ()$ ▷ Lista de tabuleiros (retorno do algoritmo).
for cada barco b em $barcos$ **do**
  for cada posição possível pos em $inicial$ **do**
    $posBarcos[b].append(pos)$ ▷ Calcula posições possíveis deste barco b.
  end for
end for
while $duracao$ **do** ▷ Neste experimento foi usado 10 segundos.
  $nt \leftarrow inicial.copia()$ ▷ Novo tabuleiro montado iterativamente.
  for cada barco b em $barcos$ **do**
    $pontuacoes = ()$
    for cada posição possível pos de b em nt **do**
      $pontuacoes.append(pontuacao(nt.com(b)))$ ▷ Segundo equação 1
    end for
    $posicaoEscolhida = sorteioPonderado(posicoes, pontuacoes)$
    $nt \leftarrow nt.com($barco na $posicaoEscolhida)$
  end for
  if $nt.valido$ **then**
    $resultado.append(nt)$
  end if
end while
return $resultado$

uma constante. Para cada sonar já disparado no tabuleiro podemos fazer esta verificação, aumentando o valor linearmente para cada sonar.

Por fim, com base na quantidade de restrições atendidas, o cálculo da função de pontuação é dado por:

$$P = 1 + 5D + 4S \ , \tag{1}$$

onde D representa o número de coordenadas danificadas atendidas pelo barco nesta posição e S representa o número de sonares que estão com valores coerentes com o barco nesta posição.

O peso diferente para cada tipo de restrição indica o quanto cada tipo de restrição traz de informação, e o quanto deve ser priorizada em relação à outra. Restrições de coordenadas danificadas representam a certeza de que um barco está naquela posição, então receberam um peso maior para cada coordenada. O número de sonares por outro lado embora denote a presença de um barco, não restringe a posição específica, e portanto recebeu um peso menor. Estes valores influenciam na eficiência de busca de tabuleiros e foram obtidos experimentalmente.

Por fim, a função de pontuação é chamada para cada possível posição para o novo barco. Por ser chamada muitas vezes, seu tempo e complexidade de execução estão diretamente associadas ao número de tabuleiros encontrados. Uma função demorada retarda

Algorithm 2 Algoritmo para medir qualidade

$top_{i,t}$: Matriz com as razões de acurácia das i melhores posições no turno t.
$qtdTabuleiros_t$: Lista com a quantidade de tabuleiros retornada em cada turno.
for 1 até 100 **do** ▷ 100 experimentos foram feitos.
  $tab \leftarrow tabuleiro.aleatorio()$ ▷ Mapa aleatório com 4 barcos aleatórios.
  for t de 0 até 10 **do** ▷ Em cada experimento, 10 turnos foram jogados.
    $tabuleiros \leftarrow MonteCarlo(tab, barcos)$ ▷ Amostra os tabuleiros. (1)
    $probs \leftarrow extraiMetricas(tabuleiros)$ ▷ Calcula probabilidade de cada célula.
    $qtdTabuleiros_t.append(tabuleiros.tamanho)$ ▷ Quantidade de tabuleiros.
    for i em {1,3,5} **do**
      $acertos =$ (1 para cada coordenada em probs(1...i) contendo barco)
      $top_{i,t}.append(soma(acertos)/i)$ ▷ Armazena a acurácia.
    end for
    $posicaoMaisProvavel \leftarrow probs[1]$
    $revelarSonar \leftarrow Chance(20\%)$ ▷ Revela um sonar com 20% de chance.
    if revelarSonar **then**
      $tab.revelarSonar(posicaoAleatoria)$ ▷ Atualiza o tabuleiro.
    else
      $tab.revelar(posicaoMaisProvavel)$ ▷ Atualiza o tabuleiro.
    end if
  end for
end for
$imprimeResultados(top, qtdTabuleiros)$

a inserção de um barco no tabuleiro atual, o que compromete o número total de tabuleiros encontrados. Dessa forma, é fundamental que esta função seja computacionalmente eficiente.

2.7. Metodologia de avaliação

Para medir a qualidade do algoritmo MCTS desenvolvido, foram simuladas cem partidas independentes do jogo. Para cada partida, quatro barcos com formatos escolhidos aleatoriamente foram selecionados, e dispostos de maneira também aleatória no tabuleiro. Estas informações ficam escondidas para o algoritmo, e servem como gabarito para simular o jogo e avaliar os resultados.

O algoritmo MCTS é utilizado para determinar a chance de barco em cada posição oculta do tabuleiro. A partir do conjunto de tabuleiros possíveis retornados pelo algoritmo MCTS, as probabilidades de barco de cada posição do tabuleiro é calculada. Por exemplo, se uma posição continha barco em 80% dos tabuleiros retornados, sua probabilidade dada por 80%. Para simular um jogador, a cada turno consideramos que com 20% de chance o jogador aciona um sonar escolhido aleatoriamente entre os sonares disponíveis (caso tenha sonar disponível). Com 80% de chance, o jogador faz um disparo na coordenada de maior probabilidade calculada.

Após a jogada, a informação do tabuleiro é atualizada (dados do sonar, ou barco ou água no caso de disparo) e temos então um novo tabuleiro. O algoritmo MCTS é então executado para este novo tabuleiro, e o processo se repete por 10 turnos (critério de parada

escolhido arbitrariamente).

Em cada turno, as posições mais prováveis calculadas pelo MCTS (top-1, top-3 e top-5) são comparadas com o tabuleiro gabarito para determinar a razão de quantas delas realmente continham um barco (e.g. das top-3 coordenadas mais prováveis, duas continham barco, logo a razão é de dois terços). Estas razões representam a acurácia do algoritmo naquele turno. Repare que cada turno fornece uma informação adicional para o algoritmo. Informações adicionais permitem melhor estimar a posição dos barcos, e intuitivamente aumentam a acurácia daquele turno. Nos experimentos cada partida foi simulada por dez turnos.

Outra métrica secundária contabilizada é a quantidade de tabuleiros diferentes obtida pelo algoritmo ao final de um tempo fixo de execução. No começo de uma partida, onde menos informações estão presentes, uma quantidade mais elevada de tabuleiros distintos é encontrada. Conforme mais coordenadas são reveladas, o número de tabuleiros distintos encontrados diminui, pois é mais difícil encontrar tabuleiros consistentes.

O algoritmo 2 apresenta o código do experimento descrito acima que será usado para avaliar a qualidade do algoritmo MCTS.

3. Resultados

A figura 5 apresenta o resultado obtido com o experimento, mostrando a média amostral (100 partidas) da acurácia do MCTS para as posições top-1, top-3, e top-5 ao longo dos turnos do jogo. Note que mesmo sem nenhum turno jogado (nenhuma informação obtida), o algoritmo não desempenha mal. Um mapa tem em média mais de 50 coordenadas que não são pedras. Uma frota de quatro barcos ocupa em média 15 coordenadas. Então, uma escolha aleatória e uniforme de posição do tabuleiro teria em torno de apenas 30% de acurácia. Mas vimos que a acurácia do top-1 é quase 45%. A segunda conclusão é que com apenas algumas informações sobre o tabuleiro (alguns turnos) o algoritmo MCTS já consegue obter uma acurácia maior do que 65%. Por último, vemos que conforme mais turnos são jogados, mais assertivo é o método, potencializando o resultado. Notamos também que o top-1 possui acurácia maior que o top-3, que possui acurácia maior que top-5. Este resultado é interessante pois mostra que o algoritmo MCTS é melhor em estimar uma única posição com barco. Isto ocorre pois uma vez estimada a posição mais provável, as outras seguintes mais prováveis não necessariamente irão corresponder a outros barcos, e sim a outras possibilidades de posições do mesmo barco.

O desvio padrão das 100 partidas do experimento é mostrado na tabela 1. Pode-se observar que o desvio padrão diminui quando aumentamos a quantidade de posições sendo consideradas para estimar a acurácia em um turno. Em geral, o top-5 possui um desvio padrão menor que o top-3, que por sua vez possui um desvio padrão menor do que o top-1. Entretanto, o desvio padrão em geral se mantém relativamente constante ao longo dos turnos, sendo o top-1 com o maior desvio padrão.

A figura 6 apresenta a média amostral (em 100 partidas) da quantidade de tabuleiros distintos gerados a cada turno (quantidade de informação). Conforme esperado, o número de tabuleiros encontrados em 10 segundos diminui com os turnos. Isto ocorre pois a cada turno temos mais restrições que limitam a quantidade de tabuleiros válidos que são encontrados pelo MCTS. Entretanto, após dez turnos jogados, a média ficou próxima de mil tabuleiros encontrados em 10 segundos.

	Quantidade de turnos do jogo										
Grandeza	0	1	2	3	4	5	6	7	8	9	10
Top-1	0,49	0,48	0,48	0,47	0,48	0,44	0,45	0,43	0,41	0,43	0,40
Top-3	0,30	0,30	0,29	0,29	0,30	0,32	0,32	0,32	0,30	0,29	0,29
Top-5	0,24	0,23	0,21	0,22	0,24	0,27	0,27	0,28	0,25	0,26	0,24

Tabela 1. Desvio padrão da acurácia das posições mais prováveis (top-1, top-3 e top-5).

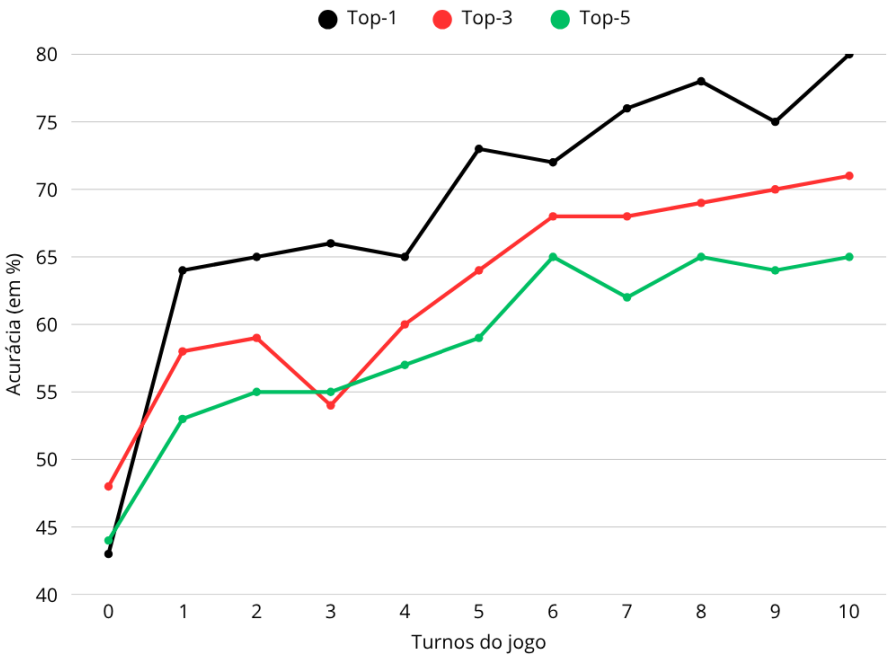

Figura 5. Média amostral da acurácia das posições mais prováveis (top-1, top-3 e top-5) ao longo dos turnos.

4. Trabalhos Relacionados

O jogo de Batalha Naval vem sendo estudado no contexto acadêmico há pelo menos 25 anos [Fiat and Shamir 1989]. Em particular, o jogo serve de inspiração para diferentes problemas em combinatória e geometria computacional. Neste contexto, dois problemas fundamentais estudados são (i) acertar um barco ainda não atingido, e (ii) afundar um barco atingido. Um terceiro problema também estudado na literatura é encontrar uma solução para uma determinada configuração de tabuleiro [Sevenster 2004].

Diversos trabalhos focam no primeiro problema considerando barcos com determinados formatos e um tabuleiro vazio e infinito [Fiat and Shamir 1989, Hainzl et al. 2022]. O objetivo é determinar um padrão de disparo mínimo que garanta que os barcos serão atingidos ao menos uma vez. Ou seja, determinar como minimizar o número de disparos (a fração de disparos, já que o tabuleiro é infinito). Resultados teóricos oferecem limitantes superiores para a fração de disparos mínima, e algoritmos ótimos para casos particulares. O segundo problema busca determinar algoritmos eficientes (em número de disparos) para afundar um barco atingido sem conhecer o padrão do barco [Crombez et al. 2020]. Neste contexto, busca-se caracterizar o número mínimo de

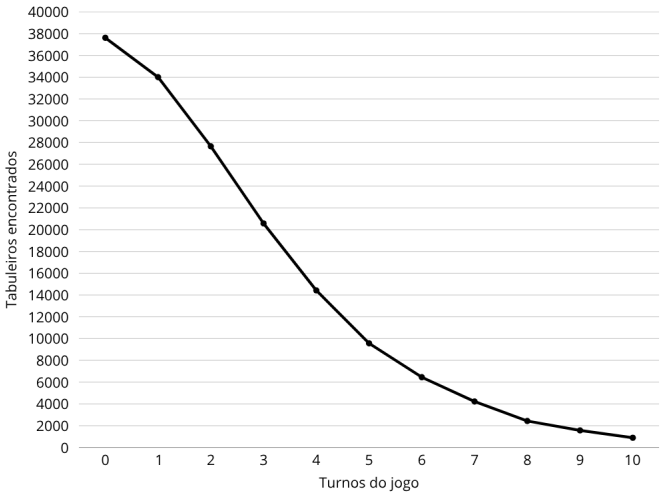

Figura 6. Quantidade de tabuleiros gerados pelo MCTS ao longo dos turnos.

disparos para afundar um barco após o primeiro acerto em função do padrão (e tamanho) do barco.

O terceiro problema trata de encontrar uma solução para uma determinada configuração de tabuleiro. Ou seja, encontrar tabuleiros que sejam consistentes com o tabuleiro atual e que contenham todos os barcos. Essa é a versão solitária (um único jogador) do jogo de Batalha Naval, pois não há rodadas e o jogador deve encontrar um tabuleiro final (com todos os barcos) consistente com o tabuleiro apresentado. Este respectivo problema de decisão é NP-Completo [Sevenster 2004], por sua conexão com o problema de empacotamento discreto no plano (*bin packing*). Neste contexto, foram propostas heurísticas para encontrar a solução para um determinado tabuleiro, utilizando algoritmos genéticos e conhecimento das pessoas (*wisdom of the crowds*) [Port and Yampolskiy 2012]. Este problema também foi representado por um modelo de programação matemática com restrições para encontrar possíveis soluções com resolvedores numéricos otimizados [Smith 2006].

Este artigo trata do problema de determinar posições do tabuleiro que sejam promissoras para o próximo disparo, sendo este um problema diferente dos estudados anteriormente. Naturalmente, o problema de encontrar um tabuleiro consistente com todos os barcos a partir de uma configuração inicial faz parte do problema aqui tratado. Por fim, a representação em árvores de modelos geométricos aqui utilizada é uma técnica antiga para estudar jogos e outros problemas geométricos [Arkin et al. 1993]. Essas árvores podem ser utilizadas por algoritmos de Monte Carlo Tree Search (MCTS) [Świechowski et al. 2023] para gerar amostras aleatórias e estimar valores de interesse, como neste trabalho.

5. Conclusão

Este trabalho apresentou uma variação do algoritmo de Monte Carlo Tree Search (MCTS) para gerar amostras de tabuleiros válidos para o jogo de Batalha Naval. A proposta atribui um peso a cada possível posição do barco a ser adicionado ao tabuleiro, fazendo com que a busca pela árvore do jogo leve a folhas que são mais prováveis de serem tabuleiros consistentes. Este peso é uma função do tipo e número de restrições que são atendidas pelo

novo barco na posição sendo considerada. Foram considerado dois tipos de restrições com pesos distintos (consistência com posições com barcos é mais importante que consistência com sonares).

O algoritmo proposto foi implementando (inclusive em um aplicativo para celular) e avaliado através de partidas jogadas com o próprio resultado do algoritmo MCTS. Em cada turno, o algoritmo MCTS foi executado por 10 segundos, para gerar amostras de tabuleiros consistentes. A acurácia das posições mais prováveis (top-1, top-3 e top-5) retornadas pelo MCTS foram estimadas em função do número de turnos jogados. A acurácia média de 100 partidas distintas indica a eficácia do MCTS, que cresce ao longo dos turnos chegando a 80% depois de 10 turnos.

Um aspecto central do trabalho é a função de pontuação para a posição de barco a ser adicionado. Esta função tem um papel crucial no comportamento do MCTS e pode levar a geração de mais ou menos tabuleiros válidos dentro de um limite de tempo (no caso, 10 segundos). Neste trabalho, foi utilizado uma heurística simples e relativamente eficiente, entretanto há muito espaço para melhorias em trabalhos futuros. Em particular, há um compromisso entre eficiência computacional e o total de tabuleiros gerados que pode ser explorado.

Referências

Arkin, E. M., Meijer, H., Mitchell, J. S., Rappaport, D., and Skiena, S. S. (1993). Decision trees for geometric models. In *ACM Annual Symposium on Computational Geometry*, pages 369–378.

Browne, C. B., Powley, E., Whitehouse, D., Lucas, S. M., Cowling, P. I., Rohlfshagen, P., Tavener, S., Perez, D., Samothrakis, S., and Colton, S. (2012). A survey of monte carlo tree search methods. *IEEE Transactions on Computational Intelligence and AI in games*, 4(1):1–43.

Crombez, L., da Fonseca, G. D., and Gerard, Y. (2020). Efficient algorithms for Battleship. In *International Conference on Fun with Algorithms (FUN 2020)*, pages 11:1–15.

Fiat, A. and Shamir, A. (1989). How to find a battleship. *Networks*, 19(3):361–371.

Hainzl, E.-M., Löffler, M., Perz, D., Tkadlec, J., and Wallinger, M. (2022). Finding a battleship of uncertain shape. In *38th European Workshop on Computational Geometry*.

Moser, Joseph M. Kramer, F. (1982). Lines and parabolas in taxicab geometry. *Pi Mu Epsilon Journal*, 7(7):441–448.

Port, A. C. and Yampolskiy, R. V. (2012). Using a ga and wisdom of artificial crowds to solve solitaire battleship puzzles. In *International Conference on Computer Games (CGAMES)*, pages 25–29. IEEE.

Sevenster, M. (2004). Battleships as a decision problem. *ICGA Journal*, 27(3):142–149.

Smith, B. M. (2006). Constraint programming models for solitaire battleships. Technical report, University College Cork, Ireland.

Świechowski, M., Godlewski, K., Sawicki, B., and Mańdziuk, J. (2023). Monte carlo tree search: A review of recent modifications and applications. *Artificial Intelligence Review*, 56(3):2497–2562.