UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
CIÊNCIA DA COMPUTAÇÃO

FLORA FRANZON BRANCHI

Meta Busca de Monte Carlo em Árvores usando Caça Níquel Multialavanca Combinatorial Com Contexto

Orientador: Prof. Dr. Anderson Rocha Tavares

Porto Alegre
2021

Dedico esse trabalho aos meus pais, que sempre me apoiam em todas as minhas jornadas

AGRADECIMENTOS

Agradeço a Universidade Federal e a a todos os funcionários e colegas que passaram pelo meu caminho e me proporcionaram tantos ensinamentos nesses anos. Agradeço especialmente ao meu amigo Ruan Leitão por ter me acompanhado e ajudado em praticamente todas as cadeiras. Agradeço também ao professor Anderson Tavares por me ajudar a conceber e a realizar esse trabalho.

SUMÁRIO

RESUMO

A técnica de Busca de Monte Carlo em Árvores (MCTS - Monte Carlo Tree Search) tem se popularizado nos últimos anos devido aos recentes resultados positivos, incluindo o desempenho sobrehumano em Go. Embora o algoritmo base de Monte Carlo tenha se mostrado efetivo em diferentes contextos, os melhores resultados são atingidos quando técnicas compatíveis com a espécie de problema são aplicados. A literatura existente apresenta uma série de estratégias que melhoram o desempenho do algoritmo quando aplicado domínios específicos, mas a determinação de qual variação é efetiva em cada domínio é atualmente feita pelo projetista, e a pesquisa se beneficiaria de uma ferramenta capaz de automatizar o processo de escolha da melhor variante do algoritmo.

Este trabalho apresenta o Meta MCTS, uma solução para o problema de seleção das melhorias do MCTS para cada problema usando Caça Níqueis Multialavanca com Contexto (CCMAB - *Contextual Combinatorial Multi-Armed Bandits*). Em CCMABs, cada contexto é um caça níquel e o objetivo é maximizar a recompensa, dada pela combinação das alavancas ativadas no caça-níquel. Em nossa modelagem, cada domínio de aplicação do MCTS é um contexto e cada possível melhoria do MCTS é uma alavanca do caça-níquel correspondente. Para contornar o problema da explosão combinatorial que ocorre com o número de melhorias, usamos a técnica de *Naïve Sampling* para definir quais melhorias selecionar em cada contexto.

Para testar o algoritmo em diferentes contextos, usamos a plataforma GVGAI (General Video Game Artificial Intelligence), onde um agente deve jogar múltiplos jogos digitais. Os resultados experimentais mostram que o Meta MCTS consegue determinar as melhorias mais promissoras para cada contexto em apenas 500 iterações de treinamento, atingindo um desempenho superior à linha de base, onde as melhorias são ligadas independentemente do contexto. Considerando-se o desempenho médio dos algoritmos para os jogos analisados, nosso método obteve resultados aproximados ao estado-da-arte.

Palavras-chave: Busca de Monte Carlo em Árvores. Caça-níquel multialavanca. General Video Game AI.

Meta Monte Carlo Tree Search using Combinatorial Multi Armed Bandits

ABSTRACT

The Monte Carlo Tree Search (MCTS) technique has become popular in the last years due to the recent positive results, including the superhuman performance in the game of Go. Even though the vanilla version of the algorithm has been successful in different contexts, the best results are achieved when strategies tuned to the specific problem are used. Existing studies showcase multiple strategies that improve the algorithm performance when used in specific domains, but the selection of the variants that are suited to a given context is currently executed by the researcher, and future projects would benefit from a tool that automates this choice of the best variant.

This work presents Meta MCTS, a solution for the selection of enhancements for each domain using Contextual Combinatorial Multi Armed Bandits (CCMABs). In CCMABs, each context is represented by a bandit and the objective is to maximize the rewards that are granted given the combination of arms activated in the bandit. In our model, each application domain is a context and each MCTS enhancement is an arm in the corresponding bandit. To deal with the combinatorial explosion that arises from the large number of enhancements, we use the Naïve Sampling technique to determine which modifications should be selected in a given context.

To test the algorithm in different scenarios we used the GVGAI (General Video Game Artificial Intelligence) framework where the agent must play multiple computer games. Our experimental results show that Meta MCTS is able to determine the most promising enhancements for each domain in only 500 training iterations, outperforming the baseline, where the modifications are used without considering the current context.

Keywords: monte carlo tree search, combinatorial multi armed bandit, general video game playing.

LISTA DE FIGURAS

LISTA DE ABREVIATURAS E SIGLAS

MCTS	Monte Carlo Tree Search
MAB	Multi Armed Bandit
CMAB	Combinatorial Multi-Armed Bandit
CCMAB	Combinatorial Contextual Multi-Armed Bandit
GVGP	General Video Game Playing
GVG-AI	General Video Game Artificial Intelligence framework
UCT	Upper Confidence for Trees

1 INTRODUÇÃO

Desde que a Busca de Monte Carlo em Árvores (BROWNE et al., 2012) foi proposta, diversos pesquisadores realizaram estudos sobre sua aplicabilidade nos mais diversos tipos de problemas: foi usado em jogos de tabuleiro, como Xadrez e Go, jogos de tempo real, como Pacman, nos mais diversos problemas de busca, incluindo o do problema do caixeiro viajante, ou até mesmo em problemas relacionados a segurança de software, sendo usado para verificar vulnerabilidades em sistemas de autenticação. Embora uma abordagem genérica consiga em alguns domínios uma performance aceitável, os melhores resultados com essa técnica foram encontrados quando usamos estratégias compatíveis diante de suas especificidades. Muitas dessas técnicas foram desenvolvidas num contexto fechado, e sua compatibilidade com outros cenários só pode ser investigada por meio de um estudo artesanal de pesquisa.

Este trabalho apresenta o Meta MCTS, uma abordagem modular na qual melhorias do MCTS podem ser ativadas ou desativadas de maneira independente. O Meta MCTS se encarrega de descobrir a seleção de melhorias adequadas ppara uma família de problemas através da modelagem do problema como um Caça-Níquel Combinatorial com Contexto (CCMAB - Contextual Combinatorial Multi Armed Bandit), onde cada contexto é representado por uma máquina caça-níquel multi alavanca específica, com suas próprias características, e o objetivo do agente é maximizar a recompensa recebida, proveniente da combinação de alavancas ativas na máquina no momento da jogada. Mais especificamente, cada caça-níquel representa um domínio de aplicação da Busca de Monte Carlo em Árvores e cada melhoria individual é uma alavanca dessa máquina. Quando tratamos de um número muito grande de alavancas teremos um problema de explosão combinatória de escolhas. Para solucionar esse obstáculo, usaremos uma técnica de amostragem chamada Naïve Sampling (ONTAÑÓN, 2013) para a seleção das alavancas a cada episódio.

Para testar o Agente diante de diferentes domínios, adotamos a plataforma GVGAI, um ambiente para desenvolvimento de algoritmos de IA generalistas. Tal plataforma disponibiliza mais de 100 jogos, cada um tratando de um domínio diferente e que responderá diferentemente às técnicas aplicadas. Os experimentos realizados com o Meta MCTS mostram que a abordagem é capaz de determinar as melhorias disponíveis que são mais compatíveis com cada jogo testado. Embora o número limitado de módulos disponíveis nessa versão não sejam suficientes para que se atinja resultados que na média são superiores aos campeões das competições do GVGAI, o aprendizado faz com que o Agente

seja capaz de ter um desempenho superior à linha base individualmente, para cada jogo, que no nosso caso é uma versão do Agente com todas as modificações desligadas. Nossa abordagem foi testada em um número reduzido de jogos, e mais testes são necessários para avaliar a efetividade em uma quantidade maior de domínios.

2 FUNDAMENTAÇÃO TEÓRICA

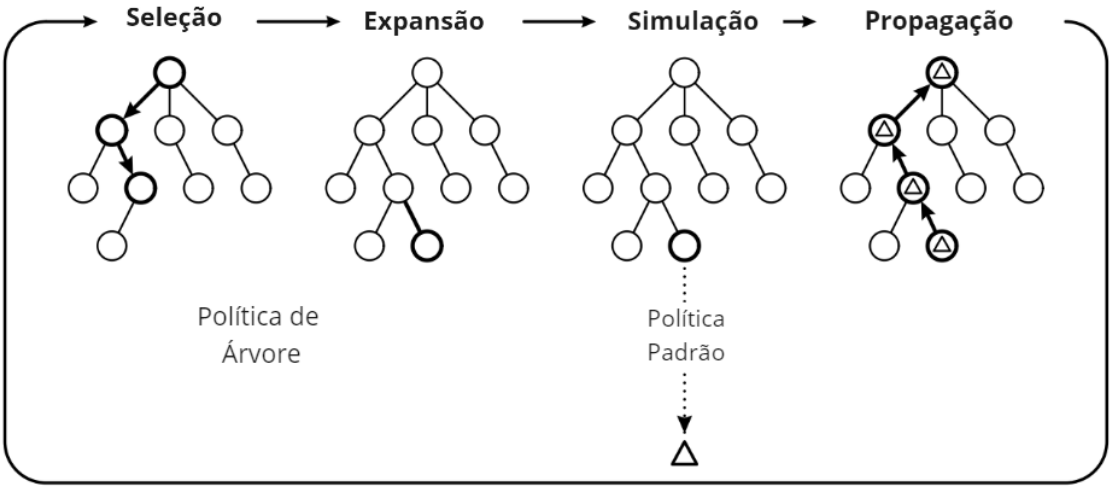
2.1 Busca de Monte Carlo em Árvores

Esta seção apresentará uma descrição básica da técnica de Busca de Monte Carlo em Árvores (MCTS - *monte carlo tree search*). Caso tenha interesse, o leitor pode consultar o survey de (BROWNE et al., 2012) onde ela é explicada em detalhes. O MCTS é um algoritmo que se baseia no princípio de que uma amostragem aleatória pode ser utilizada para obter o valor aproximado do valor dos estados. O método se popularizou muito nos últimos anos após obter sucesso sobrehumano para jogar Go - sua amostragem resolve o problema oriundo do alto fator de ramificação nas ações do jogador, onde é impossível gerar nodos para todos os possíveis estados. O método teve grande difusão na comunidade de jogos primeiramente por permitir o desenvolvimento de agentes sem conhecimento específico do domínio, sendo diretamente aplicável a qualquer domínio com o modelo generativo disponível. Esse modelo generativo permite a execução de simulações. Outro fator responsável por sua disseminação é a compatibilidade com domínios de tempo real, pois é um algoritmo anytime - ele pode ser interrompido a qualquer momento, realizando a seleção com as estimativas que estiverem disponíveis.

O MCTS se baseia na execução de um ciclo de 4 passos sequenciais - seleção, expansão, simulação e propagação - que incrementalmente geram uma árvore parcial e assimétrica, conforme Figura 2.1. O ciclo pode ser repetido durante um número definido de iterações ou enquanto houver capacidade computacional para isso. A capacidade computacional disponível pode ser relacionada a limitações de tempo. Por exemplo, em domínios de tempo real, como videogames, é necessário que exista uma cadência de ações para que o agente seja competitivo, sendo necessário definir um tempo limite para a tomada de decisão. Durante o processo, cada nodo deve guardar no mínimo 2 propriedades: o número de vezes que foi escolhido e qual a soma das recompensas obtidas na escolha. Quando os recursos são exauridos, o algoritmo retorna o que considera ser o “melhor” nodo partindo-se da raiz. A definição do que é a ”melhor” ação pode variar de acordo com a implementação, mas normalmente consiste em escolher o nodo mais visitado ou de maior recompensa.

Seleção: Durante a seleção, o algoritmo desce pela árvore a partir da raiz, utilizando uma política de árvore (tree policy) até achar um nodo que não esteja em um estado final (de derrota ou vitória) e não esteja completamente explorado, ou seja, possui algum

Figura 2.1: Ciclo de execução do MCTS. Adaptado de (BROWNE et al., 2012)



nodo filho ainda não amostrado. Pode-se pensar no passo de seleção dos nodos como sendo um problema de caça-níqueis, onde devemos escolher uma política que balanceie exploração e exploitação.

Entre as variações existentes de algoritmos do MCTS está o UCT (*Upper Confidence Bound for Trees*), proposta por (KOCSIS; SZEPESVÁRI, 2006). Essa variação, que é a mais popular dentro da familia, utiliza o UCB como política de árvore (*tree policy*), e seleciona dentre os sucessores de um nó o nodo j com a maior pontuação. A pontuação de cada nodo é calculada de acordo com a Equação 2.1, onde $\bar{X}_j$ é a média das recompensas obtidas por simulações realizadas no nodo j, *n* é o número de visitas do nodo pai, n_j é o número de vezes que o nodo *j* foi escolhido e C_p é uma constante positiva que tem impacto na taxa de exploração e depende do contexto para ser ajustada. Em caso de empates, um nodo aleatório será selecionado.

$$UCT = \bar{X}_j + 2C_p\sqrt{\frac{2\ln n}{n_j}} \tag{2.1}$$

Expansão: Durante a expansão, um nodo é adicionado à árvore a partir do nodo selecionado. Tais nodos representam o estado uma ação à frente da atual. O valor inicial de recompensa de um nodo adicionado à árvore será determinado pela simulação que é o passo a seguir.

Simulação: A partir do nodo adicionado à árvore, será realizada uma simulação, onde de acordo com alguma política de escolha de ações o estado será avançado em um número determinado de jogadas, a chamada profundidade de simulação, ou até que se alcance um estado final. A política mais comum de avanço durante a simulação é

escolha uniformemente aleatória de ações, ou seja, a cada avanço de estado escolhemos aleatoriamente entre as opções disponíveis.

Propagação: Durante o passo de propagação, o valor total da recompensa do nodo adicionado na expansão e de todos os seus predecessores será atualizado junto com número de visitas. É essa atualização imediata que define o MCTS como um algoritmo anytime, permitindo que sua execução seja interrompida a qualquer momento e a escolha da próxima ação será feita com a informação disponível.

Após a exaustão dos recursos o algoritmo é interrompido e é feita uma seleção de um dos filhos do nó raiz de acordo com alguma estratégia. O artigo de Monte Carlo (BROWNE et al., 2012) descreve 4 variações da política de seleção, sendo a primeira delas a mais comumente utilizada:

- Filho com Maior Recompensa: seleção do filho com maior recompensa média
- Filho mais Robusto: Seleção do filho com maior número de visitas
- Filho mais Robusto com Maior Recompensa: seleção do filho que tenha o maior número de visitas e a maior recompensa.
- Filho Mais Seguro: realiza a seleção de ação usando UCT

O extenso survey de (BROWNE et al., 2012) apresenta dezenas de modificações que foram propostas diante de desafios encontrados no desenvolvimento voltado a domínios, mas não trata de como elas podem ser aplicadas em outros domínios diferente do de origem. Nesse trabalho implementaremos 5 módulos que podem ser selecionados pelo Meta Agente, inspirados no trabalho de diferentes pesquisadores.

2.2 Problema do Caça-Níquel Multialavanca

O problema do Caça-Níquel (MAB - *Multi Armed Bandit*), pode ser definido por uma situação hipotética, onde um agente está diante de uma máquina caça-níquel, chamada bandit, que dá uma recompensa dependendo da alavanca (braço) selecionada. Ele espera maximizar seus ganhos num horizonte finito de jogadas, mas se depara com o dilema exploração-exploitação a cada escolha: ele pode encontrar rapidamente uma alavanca que retorna uma recompensa satisfatória, mas só descobrirá se existe outra alavanca com uma recompensa maior caso aceite o risco relacionado a experimentar uma alavanca nova e possivelmente obter uma recompensa pior do que a recompensa estimada de uma escolha conhecida. Para assegurar que a descoberta do melhor braço possível não seja

inviabilizada pela descoberta de um braço de alto valor, é necessário definir uma política de escolha em que todos os braços têm uma probabilidade não-nula de serem escolhidos.

Um exemplo de política é o ϵ-greedy, que usa uma técnica simples para balancear a exploração e o comportamento guloso dos algoritmos. Define-se um parâmetro ϵ configurável. A exploração se dará proporcionalmente a uma fração definida por 1 - ϵ. Isso significa que caso definamos $\epsilon = 0.7$, o algoritmo testará novas escolhas em 70% do tempo enquanto em 30% escolherá a melhor opção conhecida. A desvantagem do método é que como a exploração é aleatória, o segundo braço de maior valor e o de menor valor tem a mesma chance de ser escolhidos. Outros algoritmos mais complexos foram propostos devido a essa consequência, mas o ϵ greedy segue usado para tomar decisões dentro de processos mais robustos.

2.2.1 Problema do Caça-níquel Multialavanca Combinatorial

Existe uma variante do problema dos caça-níqueis conhecida como Caça-níquel Multi-alavanca Combinatorial (CMAB - *Combinatorial Multi-armed Bandit*), que foi inicialmente proposta por (GAI; KRISHNAMACHARI; JAIN, 2010), mas usaremos a formulação apresentada por (ONTAÑÓN, 2013). A diferença entre os dois tipos de problemas é que enquanto no MAB, a recompensa é dada de acordo com o acionamento de apenas uma alavanca, o CMAB cada alavanca pode estar em K posições e a recompensa é dada de acordo com a combinação de posições das alavancas. Um CMAB pode ser formalizado assim:

- Um conjunto de variáveis $X = \{X_1, \ldots, X_n\}$, onde cada variável pode assumir K valores diferentes
- Uma função de distribuição de recompensas que depende do valor de cada uma das variáveis
- Uma função que determina as combinações legais de valores de variáveis.

2.2.1.1 Naïve Sampling

A técnica de Naïve Sampling foi proposta por (ONTAÑÓN, 2013) diante de um problema detectado no seu domínio de testes, os jogos de estratégia em tempo real (RTS). Esse tipo de jogo possui um fator de ramificação muito alto em seus estados quando consideramos as ações executadas por cada unidade do jogador e cada unidade do adversário.

Para resolver o problema de explosão combinatória na seleção de alavancas que pode surgir caso tenhamos um número elevado de opções e que se agrava caso cada alavanca possa assumir mais de um valor, o autor apresenta a técnica como uma ferramenta que permite a amostragem desse tipo de espaço. Ela se baseia na premissa chamada "decomposição inocente" (2.2), que diz que podemos decompor as recompensas obtidas após a execução de uma ação descrita por um conjunto de variáveis como sendo o somatório das recompensas obtidas individualmente para cada uma das variáveis.

$$R(x_1, ..., x_n) = \sum_{i=1}^{n} = R_i(X_i) \tag{2.2}$$

Essa decomposição transforma um problema de CMAB de N variáveis em um conjunto de N + 1 problemas de MAB:

- Uma global MAB_g que possui todas as combinações de alavancas já testados até o momento. Para cada conjunto na MAB_g guardamos a recompensa média obtida ao fazer tal seleção e o número de vezes que a seleção foi feita.
- N MAB_l chamadas locais, relacionada diretamente com cada variável $\text{X}_\text{i} \in \text{X}$. Para cada uma, guardamos a recompensa média obtida para cada valor da variável e o número de vezes em que ele foi selecionado.

Com essa decomposição realizada, o algoritmo definirá a cada iteração a combinação de valores que será usada na MAB que será usada na próxima execução. O comportamento do Naïve Sampling é determinado por três políticas, que controlarão o dilema exploração-exploitação durante o processo de amostragem:

- A política π_0 define se na iteração atual o agente adotará a exploração ou exploitação.
- A política π_l é usada durante a exploração para definir o valor de cada variável.
- A política π_g é usada durante a exploitação para selecionar um MAB já amostrado.

Todas as políticas descritas costumam ser ϵ-greedy, porém com diferentes ϵ.

2.2.2 Problema do Caça-níquel Multialavanca Combinatorial com Contexto

O problema do caça-níquel combinatorial com contexto (CCMAB - Contextual Combinatorial Multi Armed Bandit) é mais uma variação do MAB, onde além das múltiplas alavancas disponíveis na máquina, temos tipos diferentes de caça-níqueis. Dessa

forma, a recompensa retornada não depende somente das alavancas selecionadas, mas também de qual máquina está sendo acionada. Na nossa proposta, o contexto é relacionado com o domínio em que o Agente está agindo, e ele deve descobrir a melhor combinação de alavancas para cada um, adotando um Naïve Sampling para cada contexto separado e guardando as pontuações obtidas e o número de vezes que cada alavanca foi acionada

3 TRABALHOS RELACIONADOS

(SOEMERS et al., 2016) realizou um trabalho extenso sobre modificações da técnica de Busca de Monte Carlo em Árvores em ambientes de tempo real. O autor questiona de que forma seria possível o uso de melhorias desenvolvidas em outras domínios para o aperfeiçoamento da performance de um agente desenvolvido no framework de GVGAI , e faz experimentos com diferentes Agentes para verificar a possibilidade de aplicação de técnicas inspiradas em trabalhos de outros pesquisadores, como por exemplo Progressive History (Nijssen and Winands, 2011) (PH), N-Gram Selection Technique (Tak et al., 2012) (NST), Tree Reuse ((PEPELS; WINANDS; LANCTOT, 2014)). Seus resultados mostram que o uso indiscriminado dessas técnicas pode causar danos à performance caso o domínio não seja compatível com o método. O uso de uma técnica chamada Loss Avoidance, que será discutida mais à frente, por exemplo, aumenta o número de simulações executadas a cada turno de jogo, diminuindo o número de nodos explorado a cada ciclo.

(BONTRAGER et al., 2016) tentou realizar um agrupamento de jogos do framework GVGAI que possuíssem características similares. Entre as características consideradas estão o determinismo, a presença ou não de inimigos e as condições de vitória. O contexto usado pelo Meta Agente em suas avaliações foi inicialmente concebido como sendo o conjunto dessas propriedades, mas conforme demonstrado pelo autor, as características escolhidas para o agrupamento não resultaram em grupos de jogos com muitas semelhanças entre si, e o uso da mesma técnica para o grupo não seria efetivo.

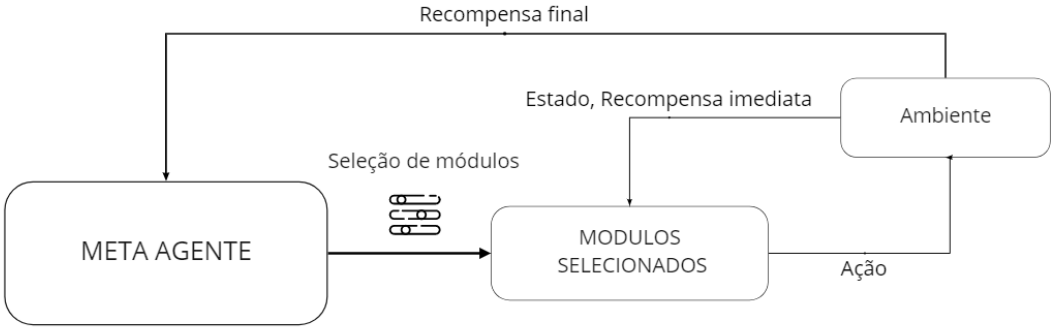
Nenhum trabalho estudado trata da transitividade de melhorias, ou seja, se elas podem ser aplicadas a um domínio diferente do que originou o seu desenvolvimento. Nosso trabalho tenta dar um passo em direção a uma inteligência generalista capaz de decidir automaticamente quais modificações são aplicáveis a domínios previamente desconhecidos.

4 META BUSCA DE MONTE CARLO EM ÁRVORES

A Busca de Monte Carlo em Árvores possui uma série de variações propostas por diversos pesquisadores de acordo com o contexto em que foram testados. A aplicabilidade de tais melhorias em domínios diferentes do domínio de origem não são conhecidas até que um estudo específico seja realizado. Propomos a criação do Meta MCTS, ilustrado na figura 4, que possui módulos que modificam o comportamento do algoritmo que podem ser ativados ou desativados individualmente a cada episódio. Seu objetivo é realizar o mapeamento do melhor conjunto de técnicas para cada problema que se deparar, sem possuir conhecimento prévio. A existência desse tipo de Agente poderia facilitar a descoberta das alterações que são transportáveis entre domínios. Esse desafio pode ser modelo como um CCMAB, onde cada contexto é um domínio, e a seleção de alavanca representa a seleção das variações ativas.

O Naïve Sampling é usado pelo Meta Agent para fazer a seleção dos módulos utilizados em cada partida. A informação dos MABs amostrados pelo Agente deve ser persistida após cada partida de acordo com o domínio aplicado, permitindo melhores escolhas conforme novos episódios de treinamento são executados.

Figura 4.1: Visão Geral do Meta MCTS



4.1 Módulos desenvolvidos para o Meta MCTS

Neste trabalho, foram desenvolvidos 5 módulos independentes, descritos a seguir: 2 deles são relacionados a variações propostas por (SOEMERS et al., 2016) e por (PEREZ et al., 2014) - Fuga de Catástrofe, Macro-Ações, respectivamente, a Pontuação Heurística é baseada num Agente desenvolvido para a competição do GVGAI de 2014, e por fim

2 deles são relacionados com decisões de implementação que fazem parte do desenvolvimento da Busca de Monte Carlo em Árvores - a utilização da inicialização prévia e a decisão sobre o nodo escolhido ao fim da execução. Os módulos serão descritos a seguir nas subseções: Pontuação Heurística (4.1.1), Pré Processamento (4.1.2), Macroações (4.1.3), Fuga de Catástrofe (**??**)e Escolha de Melhor filho (4.1.5)

4.1.1 Pontuação Heurística

Nem sempre a recompensa retornada pelo ambiente de acordo com o estado será suficiente para avaliar o quanto o agente se aproximou de seu objetivo. Quando isso acontece, devemos aplicar uma tática de *reward shaping* (ARIYUREK; BETIN-CAN; SURER, 2020) que complemente a recompensa crua provida pelo ambiente e que direcione o Agente ao comportamento desejado. Contudo, a definição de uma heurística requer conhecimentos específicos acerca do ambiente do problema.

4.1.2 Pŕe-Processamento

Em alguns domínios é possível que seja fornecido ao algoritmo um tempo inicial para a preparação antes de iniciar de fato a resolução do seu problema, e até permitindo a execução de algumas simulações. Essas simulações podem ter um efeito direto na performance pois permite que o Agente ganhe conhecimento do problema a ser resolvido. Em jogos muito ramificados, o número de ciclos do MCTS que podem ser realizados por turno é baixo, o que pode levar a um comportamento pseudoaleatório que leve a muitas derrotas. Esse módulo faz o agente usar ou não o tempo inicial disponível antes da execução.

4.1.3 Macroações

A técnica de macroações (PEREZ et al., 2015) é muito bem sucedida em domínios que exigem que a mesma operação seja repetida continuamente para que tenha um efeito relevante, como por exemplo, os jogos de plataforma, onde um agente deve pular entre as bordas das plataformas que compõem o cenário, mas antes disso deve repetir a ação de corrida para chegar mais perto, até que seja a hora correta de apertar o botão de

pulo. Essa também é uma técnica utilizada quando se trata de um domínio com espaço de busca muito complexo, pois ao repetirmos a mesma ação durante um certo tempo, diminuímos o fator de ramificação dos estados. Definimos macroação simplesmente como uma sequência de repetições da mesma ação durante M turnos.

Uma vantagem da aplicação desse artifiício é que durante o período em que a ação está definida, o Agente pode usar seus recursos para realizar um número maior de iterações antes de determinar o próximo grupo de ações. Sendo M o tamanho da macroação executada e T o tempo limite para execução dos ciclos, enquanto em um cenário de ação única o agente tem um tempo T para rodar o algoritmo, com macroações ,o primeiro grupo deve ser definido em um tempo T, mas o segundo terá $(M-1) \times T$ disponível para avaliação, resultando em uma árvore mais profunda. O espaço do problema também é reduzido, necessitando um número menor de forwards ao custo de perda de granularidade. Contudo, não é uma técnica compatível com qualquer domínio, sendo aplicável quando não há necessidade de um controle fino de ações.

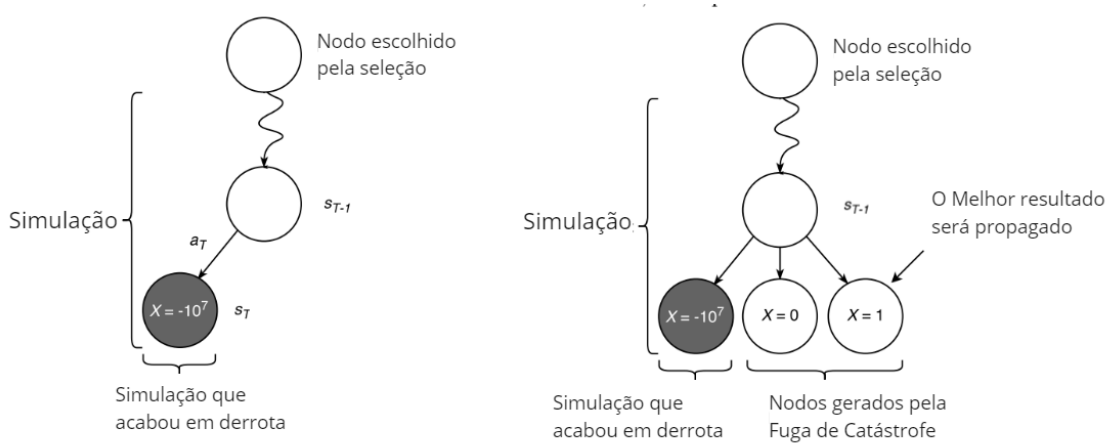
4.1.4 Fuga de Catástrofe

Fuga de Catástrofe (Loss Avoidance) é um método proposto por (SOEMERS et al., 2016) para melhorar a performance do MCTS em domínios onde há poucas situações de recompensa positiva e muitas que causam uma recompensa negativa. Um exemplo desse tipo de situação é o jogo Frogs, onde o avatar controlado pelo jogador precisa atravessar uma avenida enquanto desvia de carros. Devido a aleatoriedade na seleção de ações no passo de simulação, é provável que as simulações gerem avaliações pessimistas da situação, indicando uma derrota quando ela não aconteceria. Caso todos os nodos indiquem uma derrota, o Agente escolherá uma delas aleatoriamente, e possivelmente perderá o jogo.

O funcionamento da fuga de catástrofe ocorre da seguinte forma: durante o passo de simulação da Busca Em Árvore de Monte Carlo, caso um estado final de derrota seja atingido a partir do estado S e a ação A, o algoritmo tentará confirmar essa informação retrocedendo a simulação até o estado S_{T-1}, pai de S, e verificando se as ações irmãs de A também levariam a uma derrota. O melhor resultado obtido será então propagado pela árvore. É importante mencionar que a adição desse mecanismo potencialmente reduz o número de ciclos executados pelo MCTS antes de decidir a próxima ação, pois o uso da Fuga de Catástrofe exige que um número maior de simulações sejam nos nodos, usando

os recursos computacionais disponíveis mais rapidamente.

Figura 4.2: Funcionamento da Fuga de Catástrofe. Adaptado de (SOEMERS et al., 2016)



4.1.5 Escolha do Melhor Filho

Conforme discutido anteriormente, após o fim da exaustão dos recursos disponíveis, a Busca de Monte Carlo em Árvore deve usar alguma política para definir qual nodo filho da raiz será ser escolhido, definindo assim a próxima ação. No nosso agente teremos duas opções disponíveis: o agente pode escolher o filho de maior recompensa ou optar pelo filho que teve o maior número de visitas. O mais comum é a primeira opção, mas gostaríamos de verificar se isso teria efeito em algum experimento.

5 EXPERIMENTOS

5.1 Ambiente de experimentação GVGAI

Uma série de competições entre Agentes são realizadas anualmente ao redor do mundo, algumas contando inclusive com prêmios em dinheiro, com o intuito de incentivar a comunidade a realizar pesquisas. Estas competições são muitas vezes de jogos de tabuleiro tradicionais, como Xadrez e Go, mas recentemente com o aprimoramento das técnicas existentes surgiram competições em jogos digitais também. Contudo, as competições estão na sua maioria restritas a somente um jogo, como Mario ou PacMan, fazendo que os pesquisadores direcionem o desenvolvimento de melhorias considerando características específicas das regras do jogo e não melhorias nos algoritmos utilizados. Para incentivar as pesquisas em jogos a caminhar em direção a uma inteligência artificial genérica existem frameworks onde é possível que se defina múltiplos jogos, com regras distintas, porém com controles em comum, permitindo que a implementação de um agente jogue todos. Um deles é disponibilizado pela a GVGAI Competition, que realiza anualmente uma competição onde Agentes são testados em jogos inéditos, nunca vistos pelos competidores, forçando o desenvolvimento de técnicas genéricas aplicáveis a múltiplos domínios. A ferramenta foi desenvolvida em Java e seus agentes devem ser escritos na mesma linguagem.

Para a implementação de um Agente na GVGAI, é necessário criar uma classe que estende uma classe abstrata chamada AbstractAgent e sobrescrever 2 métodos de interação. O primeiro é o método act, que é invocado para que a próxima ação do jogo seja selecionada e o segundo é o método result, invocado quando a partida chega ao fim. Faremos nossa implementação nessa plataforma considerando sua flexibilidade e facilidade de realização testes em diferentes contextos, a fim de provar a aplicabilidade de um Meta Agente. Seguiremos as limitações de tempo determinadas para as competições oficiais ao realizar os testes, pois os trabalhos relacionados procedem dessa maneira e isso permite que seja feito um comparativo. Os limites de tempo são de até 40 millisegundos para selecionar a próxima ação e até 1 segundo para o processo de inicialização prévia. O framework permite o acesso a um objeto que descreve o estado atual do jogo chamado de StateObservation, onde podemos extrair informações sobre as entidades do jogo e usá-lo como o Forward Model para a realização das simulações. O código relacionado a esse setup experimental está disponível no GitHub, no endereço github.com/ffbranchi/GVGAI.

5.2 Implementação dos Agentes

O Meta MCTS possui as seguintes modificações para atuar no ambiente GVGAI: A primeira delas é a adição de uma ação nula que é adicionada às ações originalmente disponíveis para cada jogo. Tal ação permite que o agente opte por realizar uma jogada sem efeito, o que pode ser valioso em jogos do estilo "armadilha", onde a maioria das ações leva a uma condição de derrota. A técnica foi descrita e utilizada por alguns competidores desde a primeira competição do GVGAI (PEREZ et al., 2014) . A segunda modificação fixa é o reuso de árvore gerada. Conforme experimento realizado por (SOEMERS et al., 2016) este mecanismo aumenta o intervalo de confiança dos resultados positivos, porém não traz melhoras a resultado que tem uma performance ruim. Com a reutilização da árvore parte-se da situação onde algumas simulações já foram realizadas e a nova execução parte de uma estimativa mais real do valor dos estados. Nosso agente MCTS tentará fazer o reuso da árvore a cada nova execução caso detecte que o estado de jogo após a ação é exatamente igual à simulação gerada na jogada anterior. Variantes do mecanismo de reuso incluem uma função de decaimento para que a pontuação tenha menos valor de acordo com a idade do dado e a distância da raiz. Em nossa implementação toda a árvore é descartada caso o estado de jogo após a ação seja diferente do esperado.

O Agente também se conecta a um banco de dados SQL, onde registra as informações relacionadas a cada partida: os parâmetros usados, o jogo, o nível, o escore final e o número médio de iterações realizadas na Busca de Monte Carlo em Árvore por rodada. A simulação se mostrou a operação de maior custo computacional do processo de Busca de Monte Carlo, pois o avanço de estado do framework GVGAI consome cerca de 0,2 milisegundos, o que permite um total de 800 avanços na avaliação de cada ação em todos os nodos explorados. Para que o agente consiga realizar um número suficiente de ciclos, as simulações serão executadas com 10 jogadas de profundidade.

5.2.1 Implementação das Melhorias no GVGAI

Pontuação Heurística: Para implementar o módulo de Pontuação Heurística, nos baseamos no Agente Jin Jerry (PEREZ et al., 2014), que propõe uma função de escore calibrada pela realização de centenas de testes. JinJerry define o valor do estado baseado em observações dos objetos que compõem o jogo, como recursos, objetos que podem ser movidos, objetos estáticos, NPCs (non-player-characters) e portais. Além disso, adiciona

um termo cujo objetivo é incentivar a exploração de posições pouco visitadas no grid de jogo para evitar que o Agente fique preso em alguma parte do mapa. Uma consequência da função de avaliação escolhida é que ela estimula o Agente a fugir dos objetos presentes na cena, o que faz com que a heurística não funcione como esperado em jogos em que o jogador deve se aproximar de algo que não é considerado pela plataforma como um recurso.

Na nossa versão do Agente usamos uma versão simplificada que pudesse ser aplicável a mais jogos ao desconsiderar a pontuação oriunda dos NPCs. Os seguintes fatores fazem parte da equação de avaliação: distRecursoProximo, a distância euclidiana entre o avatar e o recurso mais próximo e distObjProximo, entre o avatar e o objeto móvel mais próximo, distPortal, entre o avatar e o portal mais próximo, visitedCount[x][y] o número de vezes que o avatar visitou aquela posição do espaço-mundo, escoreDoJogo o escore bruto e nrRecursos o número de recursos que o avatar possui naquele estado. Os valores dos termos são obtidos fazendo cálculos de acordo com as posições de estado retornadas no StateObservation do framework. É mantida também em memória uma matriz que guarda o número de vezes que cada posição foi explorada pelo jogador. As distâncias são divididas pela maior distância possível para manter os termos entre 0 e 1 e garantir que seu efeito na fórmula seja menor do que o escore bruto da plataforma. A equação que define a heurista está ilustrada na imagem 5.1

Fuga de Catástrofe O módulo desenvolvido para a Fuga de Catástrofe funciona como descrito no capítulo anterior.

Pré-Inicialização: Nas competições realizadas no framework GVGAI, é disponibilizado 1 segundo para que o construtor do Agente seja executado, então usaremos esse tempo para realizar o pré-processamento, que expandirá a árvore o quanto possivel. Optamos por parametrizar tal opção de implementação para tentar entender o impacto desse mecanismo quando usado em conjunto com outras melhorias.

Macro-ações: Para nossa implementação de macro-ações repetimos a mesma ação por 5 repetições.

5.3 Metodologia dos Testes

Para avaliar a aplicabilidade do Meta Agente descrito adotaremos uma metodologia de testes baseada na comparação entre os resultados de um Agente sem modificações e um Agente que pode escolher as modificações que usará. Foi feita uma extensão do

Figura 5.1: Descrição da Pontuação Heurística

```
Se final de jogo com vitória, escore = 1000
Se final de jogo com derrota, escore = 0
senão
      escore = escoreDoJogo
      + nrRecursos
      + 2 * escoreRecursos
      + escoreExploração
      + escoreMóveis
      + escorePortais

Sendo
escoreRecursos = 1 - distRecursoProximo / distância máxima
escoreExploração = 1 - numeroVisitas[x][y] / distância máxima
escoreObjMoveis = distObjProximo / distância máxima
escorePortais = 1 - disPortal / distância máxima
```

Agente Modular, chamado de MCTS Base, onde os módulos de melhoria estão todos desativados - neste caso, a inicialização prévia, fuga de catástrofe e macroações estão desabilitadas, o escore será diretamente extraído do GVGAI - a ação escolhida será sempre relacionada com o nodo filho de maior escore médio e não faz nenhuma inicialização prévia da árvore. A principal métrica analisada para avaliar a melhoria do Agente será a taxa de vitórias.

Primeiramente rodamos 100 partidas no Agente Base em cada jogo, sendo 20 em cada nível. A seguir, fizemos o Meta Agente treinar seus parâmetros durante 500 episódios do mesmo jogo, sendo 100 em cada nível. O software desenvolvido permite salvar os pesos dos braços do CMAB de acordo com o jogo. Durante os treinamentos, o Meta agente usou um ϵ_0 de exploração de 20%, e no CMAB o ϵ_l é de 50% e o ϵ_g é 0%. Finalizado o treinamento, executamos mais 20 partidas em cada nível carregando no início os pesos obtidos no treinamento para realizar a seleção das modificações ativas. Os agentes irão persistir os dados relativos à execução de cada partida, incluindo o valor dos 5 parâmetros flexíveis, o escore final e o número de nodos médios da árvore antes da interrupção do algoritmo, permitindo uma análise posterior. Devido às características aleatórias da Busca de Monte Carlo em Árvores, cada episódio de treinamento e execução

do Meta Agente foi repetido com os mesmos parâmetros 3 vezes. A implementação está disponível no GitHub, no repositório <http://github.com/ffbranchi/GVGAI>. Os testes foram executados em um Notebook com um processador Intel Core i7 de nona geração de 2.60GHz com 32GB de memória RAM.

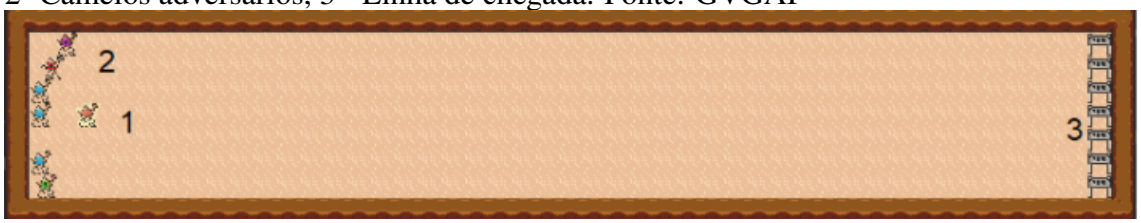
5.3.1 Jogos utilizados na Análise

Para os testes, selecionamos 3 jogos com características de jogabilidade e objetivos diferentes entre si.

5.3.1.1 Camel Race

Camel Race é um jogo de corrida em que um avatar controlado pelo jogador deve cruzar a linha de chegada antes dos camelos adversários. Esse jogo é facilmente ganho por um jogador humano, sendo necessário na fase mais simples apenas apertar repetidamente o botão de movimentação para a direita, contudo, devido ao tempo limitado de execução dos algoritmos, muitos Agentes têm dificuldade de enxergar o estado final no início da simulação e são rapidamente ultrapassados por outro corredor. O escore só é incrementado quando o camelo cruza a linha de chegada, e o uso do escore bruto do framework exige que a árvore seja expandida até o fim para que alguma ação seja relevante na avaliação.

Figura 5.2: Captura de tela do Camel Race. Legenda: 1- Camelo controlado pelo jogador, 2- Camelos adversários, 3 - Linha de chegada. Fonte: GVGAI



5.3.1.2 Crossfire

Crossfire é um jogo não determinístico onde um cavaleiro controlado pelo jogador deve desviar de balas disparadas por canhões estacionários até que atinja uma porta que finaliza o jogo com vitória. Caso ele colida com um projétil, o jogo acaba com uma derrota. Os projéteis são disparados aleatoriamente pelos canhões pelas regras do jogo, e ele às vezes se torna caótico devido a grande quantidade de balas transitando na tela, e

Figura 5.3: Capture tela do Crossfire. 1 - Avatar controlado pelo jogador, 2 - Canhão estacionário, 3 - Projétil disparado pelo canhão, 4 - Objetivo. Fonte: GVGAI

não é trivial para um humano terminá-lo com vitória.

5.3.1.3 Jaws

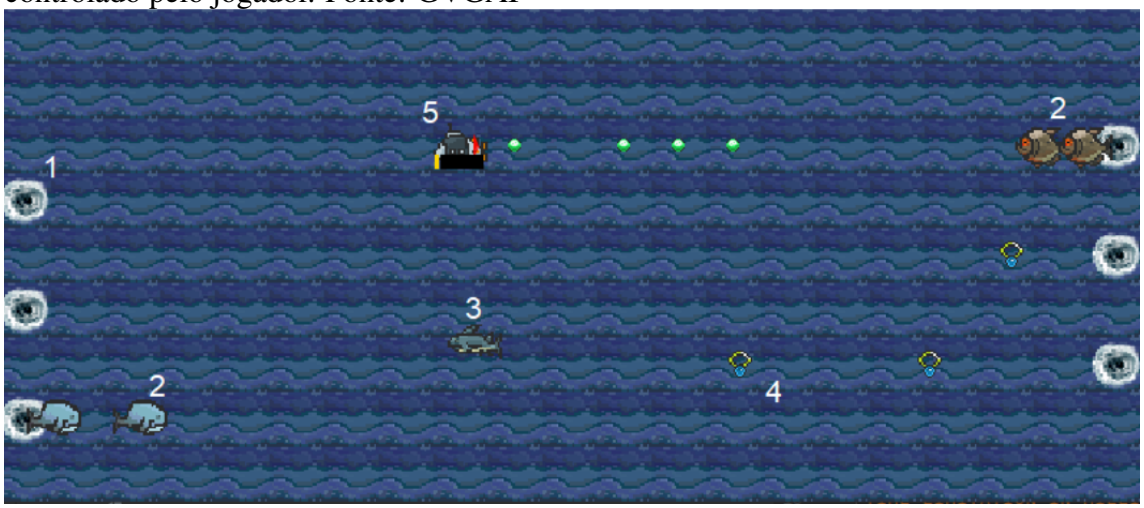
Jaws é um jogo de ação onde o jogador é uma criatura que está embaixo do mar e deve se desviar dos peixes e tubarões que surgem. É possível usar o botão de ação para atirar nos peixes a qualquer momento, e quando eles são derrotados é possível que apareça um recurso no seu lugar. Já para atirar nos tubarões um número determinado de recursos deve ser coletado. O jogo acaba em uma derrota quando o jogador colide com um inimigo e em vitória quando ele consegue sobreviver até o fim dos 2 mil turnos definidos no framework.

5.4 Resultados

5.4.1 Análise do Meta MCTS

Nos gráficos abaixo temos a curva da média da taxa de vitórias dos três repetições obtidas para cada um dos jogos, considerando uma janela de tamanho 10, ou seja, medindo o resultado nos últimos 10 jogos. É possivel verificar que o Agente ajusta sua estratégia conforme mais jogos são executados e também que em algumas situações, a técnica usada em uma fase não é aplicável à próxima, fazendo com que a taxa de vitó-

Figura 5.4: Captura de tela do jogo Jaws. Legenda: 1 - Portal de onde inimigos surgem, 2 - Peixes que podem ser imediatamente derrotados, 3 - Tubarão que exige a coleta de recursos para ser derrotado, 4 - Tesouros deixados por inimigos derrotados, 5 - O avatar controlado pelo jogador. Fonte: GVGAI



rias se reduza. Como foram executadas 100 iterações de cada fase, cada intervalo de 100 execuções representa a performance do Agente naquele nível.

5.4.1.1 Camel Race

Os testes realizados com o Camel Race indicam que o Agente é capaz de identificar uma técnica compatível com esse domínio apesar das características um pouco diferentes entre os níveis deste jogo. Enquanto é possível que o agente seja ultrapassado por outro camelo e o jogo termine rapidamente nos níveis 0, 1 e 2, no nível 3 e 4 temos barreiras que reduzem a movimentação dos adversários. Especificamente no nível 3, todos os camelos ficam presos pelo caminho e o Agente tem todo o tempo de execução do jogo para encontrar a linha de chegada. É curioso observar que apesar disso, o Agente teve 1 derrota nesse nível na segunda bateria de testes quando optou por usar o escore bruto do framework. Embora a técnica de macroações tenha sido a escolha preferencial do primeiro e segundo experimentos, no terceiro experimento essa opção foi selecionada menos de 20% das vezes, pois o Agente constatou que usando a heurística de score ele é capaz de ganhar.

Outro fator que poderia ter influenciado nos resultados é a configuração da duração da mesma macroação. Quando aplicamos macroações temos uma perda na granularidade do movimento, impossibilitando o uso da técnica quando precisamos de um controle fino de movimento no contexto. Coincidentemente, a seleção de um grupo de macroações de

Figura 5.5: Taxa de Vitória média nos experimentos com Camel Race

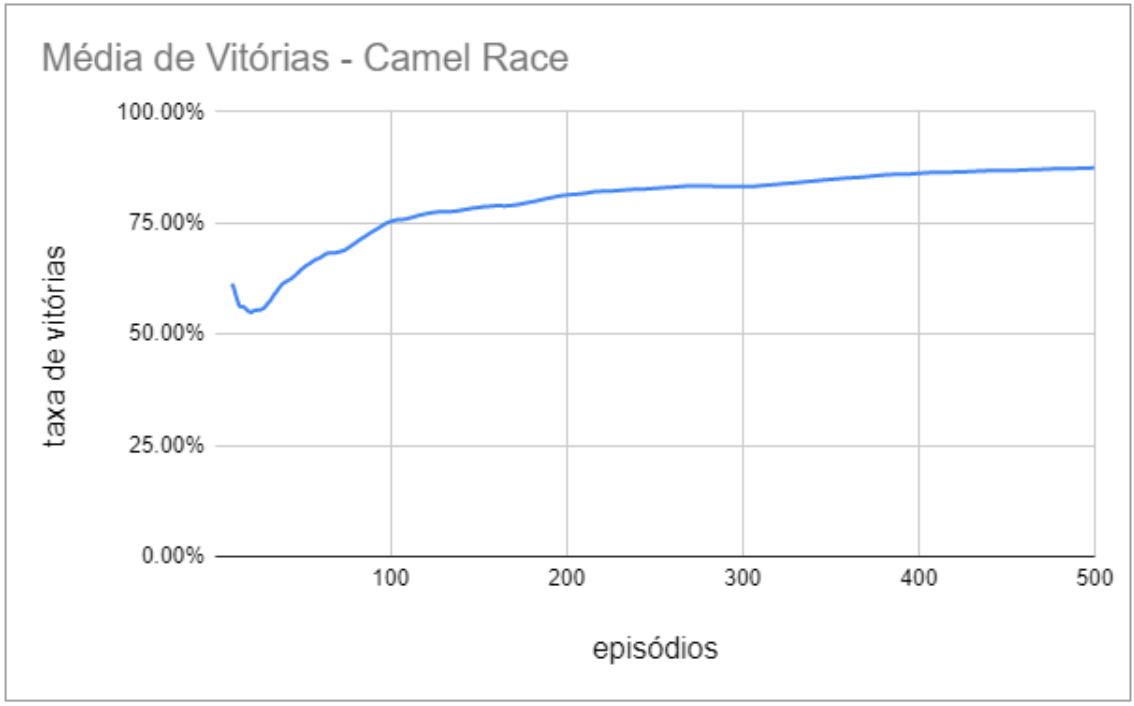


Figura 5.6: Taxa de Vitória média nos experimentos com Crossfire

Figura 5.7: Taxa de Vitória média nos experimentos com Jaws

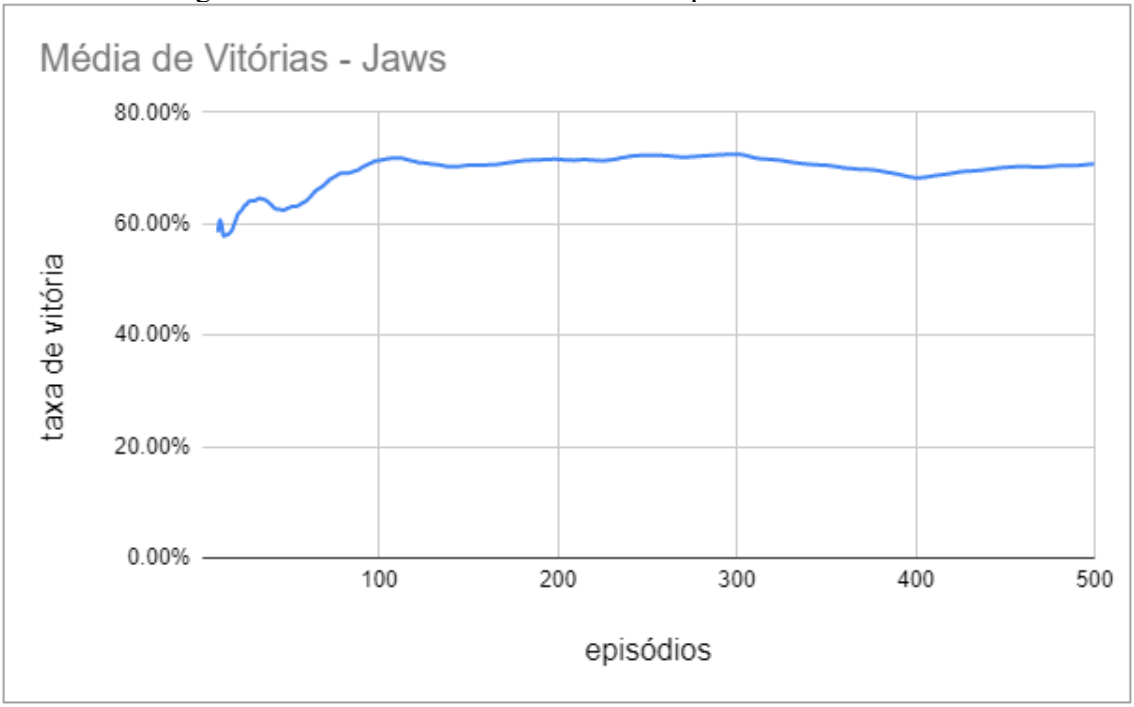


Tabela 5.1: Tabela de média de Vitórias por Nível. Em negrito, o melhor resultado de cada linha

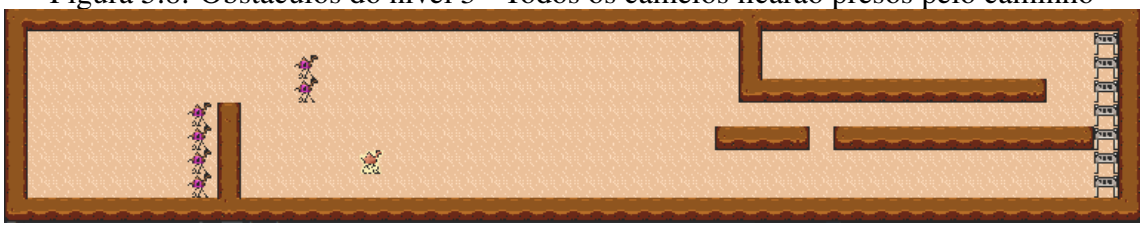
Jogo	Nivel	Percentual de Vitórias Agente Base	Percentual Médio de Vitórias do Meta Agente
Camel Race	0	0.0%	85.0%
	1	0.0%	83.3%
	2	0.0%	86.7%
	3	15.0%	98.3%
	4	0.0%	86.7%
Crossfire	0	0.0%	53.3%
	1	0.0%	48.3%
	2	0.0%	36.7%
	3	0.0%	30.0%
	4	0.0%	25.0%
Jaws	0	85.0%	71.7%
	1	95.0%	66.7%
	2	80.0%	85.0%
	3	85.0%	66.7%
	4	75.0%	78.3%

Tabela 5.2: Módulos usados pelo Agente no Camel Race. Legenda: FC: Fuga de catástrofe, MA: macroações, HE: heurística de escore, SMF: Seleção de melhor filho, PI: Pré-Inicialização

	FC	MA	HE	SMF	PI	Número de Usos
Teste 1	F	V	F	V	V	60
	F	V	F	V	F	21
Teste 2	F	V	F	V	F	71
	V	V	F	V	F	12
Teste 3	V	F	F	V	F	62
	F	F	F	V	F	11

tamanho 5 fez com que o camelo não ficasse preso nunca, embora precise eventualmente repetir alguns movimentos para desviar dos objetos e isso resulte em algumas derrotas. Caso ele ficasse trancado, possivelmente o Meta Agente não favoreceria tanto essa modificação. Para ter um Agente reativo às mudanças de nível poderíamos adicionar a fase como parte do contexto. Essa mudança inclusive se aproxima mais do comportamento humano - normalmente quando avançamos uma fase no videogame somos apresentados a novas mecânicas e perigos, e devemos ajustar nossa estratégia de acordo com a nova situação apresentada.

Figura 5.8: Obstáculos do nível 3 - Todos os camelos ficarão presos pelo caminho



Observando os módulos escolhios pelo Meta Agente na tabela 5.4.1.1 a Fuga de Derrota e Inicialização Prévia não parecem ter grande influência para este jogo, pois os resultados não foram influenciados positivamente ou negativamente por essa seleção. Também podemos observar que o uso do escore bruto proveniente do framework não leva a bons resultados, e todas as execuções optaram por usar a heurística. Por fim, podemos observar que no último experimento ele optou por não usar macroações e mesmo assim obteve bons resultados pois a heurística faz o avatar se aproximar da linha de chegada devido a presença de um portal.

5.4.1.2 Crossfire

Devido às características de aleatoriedade da amostragem da Busca de Monte Carlo em Árvores, quase todas as simulações realizadas nos nodos na versão sem mo-

Tabela 5.3: Módulos usados pelo Agente no Crossfire. Legenda: FC: Fuga de catástrofe, MA: macroações, HE: heurística de escore, SMF: Seleção de melhor filho, PI: Pré-Inicialização

	FC	MA	HE	SMF	IP	Número de Usos
Teste 1	V	F	F	F	F	75
	V	F	F	F	V	8
Teste 2	V	F	F	V	V	79
	F	F	F	V	F	5
Teste 3	V	F	F	F	V	88
	V	F	V	F	V	3

dificações para este jogo levam a um estado de derrota, fazendo com que as recompensas estimadas sejam similares e que uma ação aleatória seja escolhida muito frequentemente. É possível observar que o avatar fica preso no início da fase, executando ações randômicas. Esse comportamento causa uma taxa de vitórias de 0% nas 100 partidas executadas no Agente Base. Podemos observar os módulos mais selecionados na na tabela 5.3

O uso do mecanismo de Loss Avoidance trouxe um grande aumento na taxa de vitórias, porém só essa tática ainda coloca o Agente diante de situações onde não consegue impedir uma derrota executando alguma ação disponível. A função heurística faz com que o agente tenha um comportamento "relutante", pois ao mesmo tempo que tenta se aproximar do Portal de fim de jogo, tenta se afastar dos canhões estacionários. O problema se agrava no nível 3, 4 e 5 devido a limitação de tempo que faz com que a árvore não seja expandida o suficiente para enxergar claramente o objetivo. Para que o Agente atingisse uma performance melhor poderíamos aplicar alguma técnica que ajude a aumentar o número de nodos expandidos por ciclo.

5.4.1.3 Jaws

Jaws é um jogo com regras simples que teve a melhor performance geral no trabalho de (SOEMERS et al., 2016), já que a implementação base da Busca de Monte Carlo em Árvores já consegue atingir uma taxa de vitórias maior do que 70%. Os resultados dos experimentos mostram que diante deste cenário o uso de uma heurística não melhorou a performance pois ela não foi capaz de solucionar o motivo da maioria das finalizações de jogo. Constantemente um inimigo nasce na mesma posição do jogador quando ele se mantém em cima de um dos portais, o que causa uma derrota imediata. Os resultados mostram que nenhuma modificação implementada consegue resolver essa situação. Uma heurística específica que considerasse os locais de onde saem os inimigos perigosos e que fizesse o agente evitar essas posições poderia melhorar os resultados de jogos

Tabela 5.4: Módulos usados pelo Agente no Jaws. Legenda: FC: Fuga de catástrofe, MA: macroações, HE: heurística de escore, SMF: Seleção de melhor filho, PI: Pré-Inicialização

	FC	MA	HE	SMF	IP	Número de Usos
Teste 1	F	F	V	F	F	83
	F	V	V	F	F	3
Teste 2	F	F	V	F	V	65
	F	F	V	V	F	25
Teste 3	F	F	F	F	V	79
	F	F	V	F	V	4

que apresentam esse comportamento. As modificações selecionadas para este jogo estão disponíveis nas tabela 5.4

O Meta Agente é capaz de selecionar as modificações mais compatíveis com seu domínio que estejam disponíveis e a performance do Agente não parece ser afetada negativamente no caso de nenhuma modificação ser aplicável. Quanto maior o número de modificações disponíveis, maior a chance de uma delas ser aplicável a um novo domínio que esteja sendo estudado. A competição GVGAI coloca os agentes diante de jogos que nunca foram antes vistos, e seria interessante verificar se o Meta Agente seria capaz de fazer a seleção de alguma mudança relevante sem um grande número de iterações

5.4.2 Comparação com outras abordagens

Apresentamos também um comparativo com os resultados obtidos em pesquisas passadas para os jogos que selecionamos para o protótipo na tabela 5.5. Cada coluna é relacionada com um Agente específico, brevemente descrito abaixo:

- YBCriber: Agente vencedor da competição do GVGAI em 2015 e se baseia no algoritmo de *Iterated Width* ((GEFFNER; GEFFNER, 2015)) mas possui heurísticas específicas para o GVGAI. Ele é uma boa representação do atual estado da arte.
- SOLOMCTS: Agente disponibilizado pelos organizadores da competição GVGAI que usa a técnica de *Open-Loop MCTS*, onde somente o nodo raiz representa um estado único, enquanto os outros nodos representam a distribuição de probabilidade dos estados que são atingíveis a partir do estado inicial. (PEREZ et al., 2015)
- *Loss Avoidance*: Agente desenvolvido por Soemers que possui a modificação de Fuga de Catástrofe ativa (SOEMERS et al., 2016)
- Agente Conjunto: Agente também desenvolvido por Soemers com todas as modificações propostas pelo autor ativas. (SOEMERS et al., 2016)

Tabela 5.5: Taxa de vitórias do Meta MCTS e de Agentes utilizados na competição GV-GAI. Em negrito estão destacados os melhores resultados de cada linha

Percentual de vitórias por Agente					
Jogo	YBCriber	Agente Conjunto	MMCTS	Somente LA	SOLOMCTS
Camel Race	**100.00%**	100%	88.00%	8%	15.00%
Crossfire	**99.00%**	78%	38.67%	12%	4.00%
Jaws	7%	18%	73.67%	20%	**78.00%**
Média	**68.67** %	65%	66.78%	13%	32.33%

6 CONCLUSÃO

Neste trabalho apresentamos uma abordagem para o problema de seleção de melhorias a ser aplicadas a um agente MCTS por meio do uso de um *Contextual Combinatorial Multi Armed Bandit*, representando os diferentes domínios como diferentes caça-níqueis e cada possível melhoria como uma alavanca ativa. Para os experimentos adotamos a plataforma GVGAI, de agentes generalistas em Inteligência Artificial. Os resultados obtidos sugerem que a abordagem é promissora, pois o Agente se mostrou capaz de identificar qual modificação disponível é mais compatível com seu contexto. Porém, os testes foram realizados em um subconjunto pequeno de jogos e mais testes devem ser realizados para validar a efetividade do método em mais domínios.

Tratando-se de trabalhos futuros relacionados ao Meta Agente, um caminho promissor é verificar o comportamento do Meta Agent caso não considerássemos somente o jogo como sendo o contexto, e sim a junção de jogo e nível. Isso instintivamente se aproxima mais do comportamento humano - um novo nível de jogo costuma trazer novas funcionalidades e desafios, e o jogador deve adaptar suas técnicas existentes e possivelmente desenvolver novas estratégias para que obtenha êxito na nova situação. Outra possibilidade de extensão do Meta Agente é o uso de características mapeadas dos jogos para definir as melhorias a ser ativadas. Assim seria possível aproveitar as mesmas melhorias do MCTS em jogos diferentes, porém com algumas similaridades. A aplicação do Meta MCTS em domínios diferentes dos jogos eletrônicos também é uma questão em aberto.

Outro possível caminho é a investigação acerca do efeito de ter melhorias parametrizadas. Por exemplo, a melhoria de macroações poderia ser parametrizada pela duração da mesma ao invés de apenas ser ativada ou desativada, permitindo a investigação do efeito de macroações de diferentes tamanhos no mesmo cenário.

REFERÊNCIAS

ARIYUREK, S.; BETIN-CAN, A.; SURER, E. Enhancing the Monte Carlo Tree Search Algorithm for Video Game Testing. **IEEE Conference on Computatonal Intelligence and Games, CIG**, v. 2020-Augus, p. 25–32, 2020. ISSN 23254289.

BONTRAGER, P. et al. Matching Games and Algorithms for General Video Game Playing. **Twelfth Artificial Intelligence and Interactive Digital Entertainment Conference**, p. 122–128, 2016.

BROWNE, C. B. et al. A survey of Monte Carlo tree search methods. **IEEE Transactions on Computational Intelligence and AI in Games**, v. 4, n. 1, p. 1–43, 2012. ISSN 1943068X.

GAI, Y.; KRISHNAMACHARI, B.; JAIN, R. Learning multiuser channel allocations in cognitive radio networks: a combinatorial multi-armed bandit formulation. **2010 IEEE Symposium on New Frontiers in Dynamic Spectrum, DySPAN 2010**, n. May 2014, 2010.

GEFFNER, T.; GEFFNER, H. Width-based planning for general video-game playing. **Proceedings of the 11th AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment, AIIDE 2015**, v. 2015-Novem, p. 23–29, 2015.

KOCSIS, L.; SZEPESVÁRI, C. Bandit based Monte-Carlo planning. **Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)**, v. 4212 LNAI, n. May 2014, p. 282–293, 2006. ISSN 16113349.

ONTAÑÓN, S. The combinatorial Multi-armed Bandit problem and its application to real-time strategy games. **Proceedings of the 9th AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment, AIIDE 2013**, p. 58–64, 2013.

PEPELS, T.; WINANDS, M. H.; LANCTOT, M. Real-time monte carlo tree search in Ms Pac-Man. **IEEE Transactions on Computational Intelligence and AI in Games**, IEEE, v. 6, n. 3, p. 245–257, 2014. ISSN 1943068X.

PEREZ, D. et al. Open loop search for general video game playing. **GECCO 2015 - Proceedings of the 2015 Genetic and Evolutionary Computation Conference**, p. 337–344, 2015.

PEREZ, D. et al. Solving the physical traveling salesman problem: Tree search and macro actions. **IEEE Transactions on Computational Intelligence and AI in Games**, v. 6, n. 1, p. 31–45, 2014. ISSN 1943068X.

SOEMERS, D. J. et al. Enhancements for real-time Monte-Carlo Tree Search in General Video Game Playing. **IEEE Conference on Computatonal Intelligence and Games, CIG**, v. 0, 2016. ISSN 23254289.