Universidade Federal de Juiz de Fora
Instituto de Ciências Exatas
Bacharelado em Sistemas de Informação

Um Controlador Geral para Jogos Eletrônicos via Árvores de Busca Monte-Carlo

Eduardo Hauck dos Santos

JUIZ DE FORA
6, 2017

Um Controlador Geral para Jogos Eletrônicos via Árvores de Busca Monte-Carlo

Eduardo Hauck dos Santos

Universidade Federal de Juiz de Fora
Instituto de Ciências Exatas
Departamento de Ciência da Computação
Bacharelado em Sistemas de Informação

Orientador: Heder Soares Bernardino

JUIZ DE FORA
6, 2017

Um Controlador Geral para Jogos Eletrônicos via Árvores de Busca Monte-Carlo

Eduardo Hauck dos Santos

MONOGRAFIA SUBMETIDA AO CORPO DOCENTE DO INSTITUTO DE CIÊNCIAS EXATAS DA UNIVERSIDADE FEDERAL DE JUIZ DE FORA, COMO PARTE INTEGRANTE DOS REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE BACHAREL EM SISTEMAS DE INFORMAÇÃO.

Aprovada por:

Heder Soares Bernardino
Doutor em Modelagem Computacional

Victor Stroele de Andrade Menezes
Doutor em Engenharia de Sistemas e Computação

Saulo Moraes Villela
Doutor em Engenharia de Sistemas e Computação

JUIZ DE FORA
29 DE 6, 2017

Resumo

A área de General Video Game Playing (GVGP) propõe como desafio a criação de controladores capazes de jogar jogos eletrônicos com regras previamente desconhecidas. O desenvolvimento na área tem sido fomentado pela realização de competições envolvendo GVGP, e novos métodos têm sido propostos e testados nessas competições. Uma das técnicas mais recorrentes é a busca em árvores Monte-Carlo (*Monte Carlo Tree Search*, ou MCTS). Este trabalho investiga o desempenho de um controlador baseado em MCTS, identificando suas principais características e propondo modificações para alguns de seus pontos negativos. As modificações foram implementadas e testadas tanto em jogos de jogador único quanto em jogos de multi jogador. Os resultados indicaram que as modificações propostas foram capazes de melhor o desempenho do algoritmo para casos particulares, sem prejudicar o desempenho geral do algoritmo para os demais casos.

Palavras-chave: Inteligência Computacional, Controlador Geral, General Video Game Playing, busca em árvore Monte-Carlo.

Abstract

General Video Game Playing (GVGP) propose as a challenge the creation of controllers capable of playing video games with previously unkown rules. The development in the field has been fomented by competitions involving GVGP, and new methods have been proposed and tested on those competitions. One of most recurrent techniques is the Monte Carlo Tree Search (MCTS). This work investigates the performance of a MCTS based controller, identifying its main characteristics and proposing modificações for some of its downsides. The modificações were implemented and tested in both single player games and multiplayer games. The results indicated that the proposed modifications were able to improve the performance of the algorithm for particular cases, without hurting the general performance for other cases.

Keywords: Computational Intelligence, General Controller, General Video Game Playing, Monte Carlo tree search

Agradecimentos

Agradeço primeiramente ao meu orientador Heder, que foi o principal apoiador e incetivador deste projeto.

Agradeço aos meus pais e familiares, pelo apoio e sustento ao longo dos anos em que estive nessa universidade.

Agradeço aos amigos e colegas que estiveram em algum momento ou durante todo o caminho comigo nesta jornada.

Finalmente, agradeço a todos os professores e funcionários da UFJF que direta ou indiretamente tenham feito parte dessa conquista.

Conteúdo

Lista de Figuras **6**

Lista de Tabelas **7**

Lista de Abreviações **8**

1 Introdução **9**
1.1 Justificativa . . . 11
1.2 Objetivos . . . 11
1.3 Metodologia . . . 12
1.4 Organização do texto . . . 12

2 Revisão Bibliográfica **14**
2.1 Agentes e controladores em jogos . . . 14
2.2 General Game Playing . . . 15
2.2.1 Técnicas comuns em controladores gerais . . . 16
2.2.2 Comparações entre técnicas . . . 17

3 Framework GVG-AI **19**
3.1 Linguagem de descrição de jogos . . . 19
3.2 VGDL . . . 20
3.3 java-VGDL . . . 22
3.3.1 Elementos do jogo . . . 22
3.3.2 Estados . . . 23
3.3.3 Ações . . . 23
3.3.4 Forward model . . . 24
3.4 Modos de jogo . . . 24
3.4.1 Quantidade de jogadores . . . 25
3.4.2 Tipo de física . . . 25

4 Monte Carlo Tree Search **26**
4.1 Estratégia de seleção . . . 27
4.2 Estratégia de simulação . . . 28
4.3 Política de recomendação . . . 29
4.4 *Open loop vs. Closed loop* . . . 29
4.5 Características do algoritmo MCTS . . . 30
4.6 Desvantagens do algoritmo MCTS . . . 31

5 Melhorias propostas **32**
5.1 Desencorajar ações redundantes . . . 32
5.2 Evitar ações de derrota . . . 34

6 Experimentos e análise **36**
6.1 Resultados . . . 37
6.1.1 Jogador único . . . 37
6.1.2 Dois jogadores . . . 41

7 Conclusão e Trabalhos Futuros **46**

Referências Bibliográficas **47**

Lista de Figuras

3.1 Código VGDL para o jogo Butteflies. 21

4.1 Representação do algoritmo MCTS (adaptado de CHASLOT *et al.* (2008)) . 27

5.1 Pseudo-código para o algoritmo de identificação de ações redundantes. . . . 34

6.1 Resultado final da competição 2-P GVG-AI 2017. 44

Lista de Tabelas

6.1 Jogos de jogador único selecionados para fazer parte do conjunto de testes. Legenda: D - Determinístico, E - Estocástico 36
6.2 Resultado da execução dos jogos. O controlador com a maior porcentagem de vitórias para cada jogo está marcada em negrito. O valor entre parênteses indica a pontuação média e o desvio padrão 38
6.3 Quantidade média de simulações a cada passo de tempo para cada jogo. . . 40
6.4 Quantidade média de ações redundantes por simulação. 41
6.5 Comparação de resultados entre o MCTS vanilla e o controlador submetido. Legenda: Cp - Competitivo, Cl - Colaborativo 42
6.6 Comparação de resultados entre o MCTS vanilla e o controlador submetido. 43

Lista de Abreviações

DCC	Departamento de Ciência da Computação
UFJF	Universidade Federal de Juiz de Fora
IA	Inteligência Artificial
GGP	General Game Playing
GVGP	General Video Game Playing
AAAI	Association for the Advancement of Artificial Intelligence
MCTS	Monte Carlo tree search
AG	Algoritmo Genético
UCT	Upper Confidence bounds applied to Trees
GVG-AI	General Video Game AI Competition

1 Introdução

A área de Inteligência Artificial (IA) tem proporcionado diversos desafios aos pesquisadores com o objetivo de criar máquinas com comportamento inteligente similar ao dos seres humanos (RUSSELL *et al.*, 2003).

A área de jogos tem fornecido uma incrível plataforma de testes para pesquisas em IA através da criação de agentes capazes de resolver desafios complexos em jogos. Nas últimas décadas, novas técnicas em inteligência computacional têm sido propostas na implementação de agentes para diferentes jogos. Em especial, jogos tradicionais de tabuleiro como Xadrez e Damas são alvos comuns dos pesquisadores, por apresentarem um número grande de possíveis estratégias diferentes a serem exploradas durante uma partida (RUSSELL *et al.*, 2003).

Entretanto, esses métodos são normalmente projetados para lidar apenas com um domínio específico de conhecimento, não sendo capazes de resolver desafios fora do conjunto de regras definidas para o qual foram desenvolvidos. Quando busca-se criar programas de comportamento inteligente similar ao dos seres humanos, torna-se necessária a criação de agentes capazes de jogar jogos com regras previamente desconhecidas. Neste sentido, o desenvolvimento de agentes capazes de generalizar sua tomada de decisão, também conhecidos como controladores gerais para jogos, tornou-se tema de interesse na área de inteligência artificial.

O termo *General Game Playing* (GGP) foi adotado na descrição de técnicas de inteligência computacional que permitam a um controlador jogar jogos de regras diferentes, sendo mais tarde estendido para *General Video Game Playing* (GVGP), para abranger também aos vídeo games (LEVINE *et al.*, 2013). O interesse e o desafio oferecido por essa área motivou o surgimento de competições (GENESERETH *et al.* (2005); PEREZ *et al.* (2015A)) onde os participantes enviam seus controladores, que são colocados para jogar um conjunto de diferentes jogos desconhecidos previamente.

Para que um controlador geral possa se adaptar às regras de um determinado jogo e resolver os desafios propostos pelo mesmo, é necessário que o controlador tenha acesso

a alguma informação sobre o jogo, como por exemplo, os possíveis comandos de entrada e o estado resultante gerado por um determinado comando. Através destas informações, pode-se criar uma função de avaliação para o estado atual e uma ação adequada a ser tomada em determinado momento pode ser selecionada. Entretanto, diversos desafios surgem no momento de elaborar um controlador GVGP.

Primeiramente, há um limite de tempo computacional bastante rígido a ser respeitado, já que vídeo games operam em tempo real e podem exigir uma ação a ser executada a cada fração de segundo. Isso significa que determinados métodos gulosos como a busca em largura, que exploram o espaço de busca por inteiro (alta diversificação) mostram-se eficientes apenas para alguns problemas específicos (SOEMERS *et al.*, 2016), não possuindo uma boa capacidade de generalização.

Além disso, o uso de heurísticas específicas é desaconselhável neste tipo de ambiente (PEREZ *et al.*, 2016). Por exemplo, em um jogo de vídeo game comum, guiar o agente em direção a um item e atacar os oponentes no caminho pode ser uma boa estratégia. Entretanto, para um ambiente GVGP as regras do jogo são previamente desconhecidas. Ao utilizar heurísticas específicas, um determinado controlador pode fornecer ótimos resultados para um determinado jogo, mas provavelmente sofrerá uma queda no desempenho em jogos em que as regras introduzam novas condições ou situações para o jogador. Portanto, escolher uma abordagem que possua melhor generalização torna-se um desafio.

Para obter melhorias para as técnicas analisadas, ou mesmo a utilização de novas técnicas de inteligência computacional, pode-se partir dos resultados fornecidos pelos métodos propostos na literatura e identificar classes de problemas que estes métodos não são capazes de resolver.

Este trabalho partirá de uma técnica popular para IA em jogos: o algoritmo de busca em árvores Monte Carlo (do inglês, *Monte Carlo tree search*, ou MCTS). O uso deste algoritmo é recorrente em competições GVGP, e as melhores posições são normalmente alcançadas por controladores baseados nesta técnica (PEREZ *et al.*, 2016).

1.1 Justificativa

A área de GVGP foi proposta recentemente e busca criar agentes autônomos capazes de generalizar sua tomada de decisão. Como os vídeo games oferecem uma forma de simulação em menor escala de diversas situações existentes no mundo real, pode-se ver a área de GVGP como uma aproximação da área de inteligência artificial geral (também conhecida como IA Forte), que busca criar máquinas com potencial de executar qualquer tarefa intelectual capaz de ser executada por seres humanos (BANSAL, 2012).

Embora os métodos propostos até então na literatura sejam utilizados em jogos, na sua maioria, simples e bi-dimensionais, espera-se que os estudos das técnicas de inteligência computacional possam ser expandidos para jogos cada vez mais complexos. Portanto, os resultados de pesquisas nesta área podem trazer contribuições tanto para a área de inteligência artificial como para a própria indústria de jogos eletrônicos.

PEREZ *et al.* (2016) levantou os principais desafios e oportunidades que surgiram na área de GVGP com o estabelecimento da competição GVG-AI. Entre outras questões, é mencionado que nenhuma das técnicas propostas na criação de controladores foi capaz de dominar as restantes. De fato, a média de jogos vencidos pelos diferentes ganhadores entre todas as edições da competição era de apenas 50%, ou seja, os melhores controladores perdiam, aproximadamente, 1 em cada 2 jogos. Esse é um indicativo de que o desafio proposto pela área de GVGP ainda está longe de ser resolvido.

O estudo e a comparação do desempenho de diferentes técnicas de inteligência computacional na criação de controladores GVGP pode ajudar os pesquisadores a identificarem as características, vantagens e desvantagens de cada técnica, permitindo que seus esforços sejam direcionados a abordagens que se mostrem mais promissoras.

1.2 Objetivos

Propor e avaliar melhorias para o algorítimo de busca em árvores Monte Carlo (MCTS) na criação de controladores GVGP. Entre os objetivos específicos, estão:

- Identificar as principais características do algoritmo MCTS para o problema de *General Video Game Playing*.

- Propor melhorias para o algoritmo original.
- Implementar e testar as melhorias propostas em um dois conjuntos de jogos: um de jogador único e outro de dois jogadores.
- Analisar e avaliar comparativamente os resultados obtidos pelo algoritmo com as melhorias em relação ao algoritmo original a partir dos dados de execução.

1.3 Metodologia

Para apoiar o desenvolvimento das atividades, o *framework* da competição GVG-AI (PEREZ *et al.*, 2015A) será utilizado. Este *framework* é disponibilizado em código aberto[1], oferecendo os principais recursos necessários para a implementação e execução de controladores GVGP e um grande número de jogos diferentes já implementados.

Para garantir que os resultados apresentados neste trabalho possam ser comparados com outras publicações da área, a forma de avaliação dos controladores seguirá o mesmo padrão adotado por outros trabalhos na área (FRYDENBERG *et al.* (2015); SOEMERS *et al.* (2016); PEREZ; SAMOTHRAKIS; LUCAS (2014), entre outros): executando-se o controlador por um determinado número de partidas para cada jogo, para um conjunto fixo de jogos. As comparações serão feitas avaliando a porcentagem total de vitórias e a média de pontuação para cada jogo. Além disso, os métodos propostos também serão avaliados via participação na competição *GVG-AI* na modalidade multijogador.

1.4 Organização do texto

Este trabalho é estruturado em 6 capítulos. O capítulo inicial forneceu uma introdução geral ao tema de *General Video Game Playing* e o capítulo seguinte apresenta o referencial teórico, permitindo ao leitor familiarizar-se com o tema e entender o desenvolvimento obtido na área até então. O terceiro capítulo traz uma descrição detalhada do *framework* adotado para a realização de experimentos neste trabalho. No quarto capítulo, o algoritmo

[1] https://github.com/EssexUniversityMCTS/gvgai

de busca em árvores Monte Carlo será descrito. No quinto capítulo, a metodologia e os resultados dos experimentos serão apresentados em detalhes, e uma análise dos resultados será realizada. O capítulo final apresenta a conclusão e sugestões de trabalhos futuros.

2 Revisão Bibliográfica

2.1 Agentes e controladores em jogos

Os jogos sempre se mostraram uma fonte valiosa de entretenimento para a humanidade. Eles são capazes de desafiar intelectualmente os jogadores, abstraindo competições reais (como um esporte ou uma guerra) ou mesmo competições arbitrárias idealizadas pelo criador do jogo. Especialmente no caso dos jogos de tabuleiro, o estado do jogo pode ser representado computacional e há uma quantidade limitada de ações a serem executadas pelo jogador ou agente em um determinado momento, o que facilita a simulação do jogo em um ambiente computacional. Por conta disso, os jogos de tabuleiro se tornaram um alvo de estudo frequente pelos pesquisadores de Inteligência Artificial.

De fato, os jogos se tornaram uma das primeiras áreas a serem pesquisadas em Inteligência Artificial. Em 1950, os primeiros programas de xadrez já começavam a ser desenvolvidos por Claude Shannon e Alan Turing (RUSSELL *et al.*, 2003).

SAMUEL (1959) descreve um dos primeiros programas capazes de competir com jogadores amadores de Damas. O agente então proposto realiza a busca em uma árvore onde o nó raiz representa o estado atual do jogo, e cada nó filho representa uma ação possível de ser realizada. Uma pontuação é calculada para cada sequência de ações e a ação pertencente à sequência de ações com a melhor pontuação é executada.

Desde então, um progresso crescente tem sido observado nessa área. Em 1989, o controlador Chinook começou a ser desenvolvido na Universidade de Alberta, tornando-se o campeão mundial de Damas em 1994 (RUSSELL *et al.*, 2003). Em 1997, o controlador Deep Blue desenvolvido pela IBM venceu o campeão mundial de Xadrez, Garry Kasparov (CAMPBELL *et al.*, 2012). Ainda mais recente, em 2016, o controlador AlphaGo desenvolvido pela Google Deepmind foi capaz de vencer o campeão mundial de Go, Lee Sedol (SILVER *et al.*, 2016).

O desenvolvimento nessa área foi resultado tanto de melhorias nas tecnologias de computação, permitindo que as capacidades de processamento e armazenamento dos

computadores se tornassem cada vez maiores, como do estudo de novas técnicas em inteligência computacional que puderam fazer uso destes recursos computacionais.

2.2 General Game Playing

Embora o desenvolvimento de controladores cada vez melhores para jogos como Damas ou Go sejam marcos importantes na área de IA, ainda possuem valor limitado por não serem capazes de realizar tarefas fora do domínio para o qual foram especificados. Um controlador desenvolvido especificamente para um determinado jogo dificilmente será capaz de produzir resultados em um jogo com regras diferentes. PITRAT (1968) cunhou então o termo controlador de jogo geral (do inglês, *General Game Playing* ou GGP) para definir controladores capazes de vencer jogos com regras diferentes. A área de GGP busca o desenvolvimento de técnicas de Inteligência Artificial capazes de vencer jogos previamente desconhecidos sem intervenção humana.

Em 2005, pesquisadores da Universidade de Stanford estabeleceram uma competição anual de controladores gerais, a AAAI GGP, para fomentar o desenvolvimento de pesquisas nesta área (GENESERETH *et al.*, 2005). A competição consiste de duas fases: preliminares e finais. Nas preliminares, agentes são colocados em diferentes jogos e os 8 times com as melhores pontuações avançam para as finais. Para as finais, dois agentes de times diferentes são colocados para jogar um contra o outro, avançando aquele que conseguir vencer a partida. Desde o início desta competição, diversos trabalhos vêm sendo publicados com o resultado de novas abordagens na criação de controladores gerais. Algumas dessas abordagens serão mencionadas posteriormente.

De acordo com GENESERETH *et al.* (2005), na definição de GGP "[...] o ambiente é atualizado apenas em resposta às ações dos jogadores". Porém, sabe-se que em jogos eletrônicos de vídeo game frequentemente o ambiente de jogo é atualizado independente da execução de uma ação por parte do jogador. Neste sentido, LEVINE *et al.* (2013) estendeu o termo GGP para GVGP (do inglês, *General Video Game Playing*), para incluir também jogos de video game, descrevendo suas peculiaridades, desafios extras, e as condições necessários na criação de um ambiente propício à utilização de controladores. A descrição de um *framework* capaz de apoiar o desenvolvimento de pesquisas nesta área

também foi fornecida.

Uma nova competição foi criada em 2014 com o objetivo de fomentar o desenvolvimento de técnicas na criação de controladores GVGP, a GVG-AI (PEREZ *et al.*, 2015A). Posteriormente, novas modalidades de competição passaram a ser oferecidas, como a opção de dois jogadores e de geração de fases para jogos. O estudo de controladores no ambiente desta competição será o foco deste trabalho.

Durante o ano de 2017, ocorrerá uma nova edição da competição para cada uma das modalidades oferecidas. Como forma de avaliação dos métodos propostos neste trabalho, será realizada a participação na modalidade de dois jogadores desta competição, que ocorrerá em data coincidente com a produção deste trabalho.

2.2.1 Técnicas comuns em controladores gerais

A organização de competições em GGP motivou a publicação de diversos trabalhos com a descrição de técnicas utilizadas em controladores gerais. GENESERETH; BJÖRNSSON (2013) relatam alguns dos progressos obtidos nos primeiros 8 anos de competição da AAAI GGP, mais especificamente, em três áreas diferentes: heurísticas independentes, pesos de regras em heurísticas e o uso de busca em árvores Monte Carlo (do ingles, *Monte Carlo tree search*, ou MCTS).

As primeiras competições da AAAI GGP introduziram os métodos de heurísticas independentes para jogos, que incluem mobilidade (o número de ações disponíveis ao jogador), mobilidade inversa (agir contra a mobilidade do oponente) e proximidade do objetivo (similaridade entre estados intermediários aos estados finais). Entretanto, o uso dessas heurísticas só apresenta bons resultados em alguns tipos específicos de jogos. O uso de pesos nas regras mostrou-se adequado para balancear as deficiências das heurísticas independentes, atribuindo pesos a diferentes heurísticas gerais (CLUNE, 2007). Entretanto, o avanço mais significativo surgiu com a introdução do uso de busca em árvores Monte Carlo (FINNSSON; BJÖRNSSON, 2008).

Embora a utilização de MCTS apresente algumas limitações (PEREZ; SAMOTHRAKIS; LUCAS, 2014), controladores criados a partir de variações desta técnica tendem a apresentar bons resultados durante as competições. Conforme notado por CHAMPAN-

DARD (2014), três características se destacam no uso desta técnica: sua capacidade de lidar bem com a imprevisibilidade dos vídeo games, sua capacidade de evitar más decisões através da sua busca em largura e o limite que pode ser imposto no custo computacional da sua execução.

Os algoritmos genéticos (AGs) também têm sido utilizados para a implementação de controladores (GAINA *et al.*, 2017) Os AGs são uma forma particular de algoritmos evolutivos, caracterizado pela busca baseada no processo biológico de evolução natural, com elementos tais como seleção dos mais aptos, mutação e recombinação (LINDEN, 2008). Por serem utilizados na obtenção de soluções aproximadas em problemas de otimização, o uso de AGs se mostra apropriado no contexto de GVGP. Entretanto, o tempo computacional limitado pode ser um agravante no desempenho de um controlador implementado com esta técnica.

Uma abordagem alternativa é o uso de aprendizagem por reforço (ROSS, 2014). Este método também se mostra apropriado para uso em GVGP, pois o controlador pode começar sem nenhuma informação e aprender conforme o decorrer do jogo. Entretanto, o uso de técnicas de aprendizagem por reforço também oferece limitações, como sua incapacidade de lidar bem com dados incompletos de estado, problemas de escalonamento para problemas realísticos e ambientes não estacionários (MORIARTY; SCHULTZ; GREFENSTETTE, 1999). Entretanto, ROSS (2014) discute que, embora esses pontos sejam problemáticos em um contexto de GVGP, seus efeitos podem ser mitigados de acordo com a implementação da técnica.

2.2.2 Comparações entre técnicas

A realização de competições em GGP e GVGP também motivou a publicação de trabalhos comparando controladores implementados com as novas técnicas propostas durante estas competições, que serão descritas a seguir.

FRYDENBERG *et al.* (2015) investigou variações da forma original do MCTS, propondo quatro diferentes modificações no algoritmo que poderiam ser combinadas na implementação de um controlador. As modificações incluíam enviesar a função de avaliação (*MixMax backups*) para aumentar a busca pelo risco do controlador, o uso de ações

repetidas (*Macro actions*) para aumentar a profundidade da busca, o relaxamento da regra de expansão dos nós (*Partial expansion*), permitindo percorrer netos de um nó antes que todos os seus filhos tenham sido expandidos, e uma nova função de avaliação (*Reversal penalty*) que pune o controlador por explorar áreas recentemente visitadas.

Algumas dessas combinações resultaram em mais vitórias ou maiores pontuações em determinados jogos quando comparados ao MCTS original. Entretanto, foi-se observado que nenhuma dessas combinações era dominante em relação às outras, e que o melhor método a ser utilizado poderia variar de acordo com as características do jogo.

SAMOTHRAKIS *et al.* (2015) propôs quatro possíveis combinações de técnicas para aprendizagem evolutiva, avaliando a sua capacidade de aprendizagem em um ambiente de GVGP. Foi-se utilizado o algoritmo evolution *Separable Natural Evoloution Strategies*, utilizando redes neurais e função linear como aproximadores, utilizando políticas épsilon-gulosa e softmax.

Observou-se que, embora um controlador implementado com uma das combinações mencionadas tenha apresentado melhor desempenho em relação aos demais, os resultados variavam de jogo para jogo. Além disso, a abordagem utilizada se mostrou ineficiente para jogos que requerem um plano de longo prazo para chegar ao objetivo ou alcançar a vitória.

3 Framework GVG-AI

A competição GVG-AI propõe como desafio a criação de controladores a serem testados em um conjunto de jogos com regras previamente desconhecidas. Para apoiar a competição, foi desenvolvido um *framework* para processar uma linguagem capaz de descrever jogos de vídeo game chamada de VGDL. Este capítulo fornecerá uma descrição detalhada do *framework* e da VGDL.

3.1 Linguagem de descrição de jogos

Um dos requisitos básicos para que um jogador ou controlador possa tomar a melhor ação em determinado momento de um jogo é a habilidade de raciocinar de forma lógica sobre o seu estado atual. Para que isso aconteça, é necessário que haja o entendimento das regras básicas do jogo. No caso de controladores gerais, onde se desconhece previamente as regras do jogo, uma solução comum para permitir que o programa obtenha informações básicas sobre o jogo é através de uma linguagem de descrição de jogos (do inglês, *Game Description Language*, ou GDL).

LOVE *et al.* (2008) descreve uma GDL formal para representar o estado e as regras de um jogo de maneira que possam ser processados por um controlador. Desta forma, um controlador pode extrair conhecimento específico de domínio de um jogo (SCHIFFEL; THIELSCHER, 2009) e calcular uma função de avaliação apropriada (KUHLMANN; STONE, 2006; CLUNE, 2007; SCHIFFEL; THIELSCHER, 2007).

Entretanto, esta linguagem é limitada a jogos determinísticos e baseados em turnos, sendo mais apropriadamente utilizada para jogos de tabuleiro. SCHAUL (2013) propõe então a VGDL, uma linguagem de descrição focada em vídeo games, capaz de apoiar a descrição de jogos de vídeo game bi-dimensionais e uma biblioteca em Python capaz de processar e executar esses jogos. Esta linguagem de descrição passou a ser usada como base na competição GVG-AI (PEREZ *et al.*, 2015A).

3.2 VGDL

A VGDL foi projetada com o objetivo de permitir a descrição de jogos em tempo real com um avatar de jogador. PEREZ *et al.* (2015A) fornece uma descrição detalhada desta linguagem, a qual será reproduzida aqui por completude.

A VGDL propõe uma estrutura baseada em objetos que interagem em um espaço bi-dimensional. Todos os objetos se localizam em um espaço retangular com coordenadas associadas. Objetos podem se mover de maneira ativa através de ações pré-definidas (no caso de NPCs e outros objetos do jogo) e através de ações do jogador ou controlador (no caso de avatares) ou de maneira passiva, sujeito à física do jogo. Uma interação entre dois objetos é sujeita às regras definidas por estes objetos. Uma colisão pode resultar no desaparecimento, surgimento, transformação do objeto ou em alterações nas suas propriedades.

Um jogo é definido por dois componentes distintos: a descrição da fase, que define as posições iniciais de todos os objetos e o layout do jogo em 2D, e a descrição do jogo, que descreve a dinâmica e potenciais interações entre objetos do jogo. Cada jogo é finalizado após uma determinada contagem de tempo finita, com um inteiro indicando pontuação juntamente com uma indicação de vitória ou derrota do avatar.

A descrição do jogo é formada por quatro blocos de instrução: o *LevelMapping* descreve como transformar os personagens descritos na descrição da fase em um ou mais objetos para gerar o estado inicial do jogo; o *SpriteSet* define as classes dos objetos organizados em uma árvore, onde uma classe filha herda todas as propriedades da classe pai; o *InteractionSet* define os potenciais eventos de interação a serem gerados quando dois objetos colidem, mapeando dois objetos a um método de evento; finalmente, o *TerminationSet* define as diferentes formas como um jogo pode chegar ao fim.

As definições de classes que descrevem os objetos do jogo podem conter diversas propriedades que determinam como são visualizados durante o jogo (cor, forma, orientação), como são afetados por física (massa, momentum), e qual comportamento individual possuem (e.g. perseguir o avatar). Os objetos também podem ter propriedades adicionais que definem recursos como pontos de vida ou munição, que podem ser incrementados ou decrementados conforme o decorrer do jogo.

Um exemplo de código deste linguagem é dado na imagem 3.1, com a descrição do jogo *Butterflies*:

```
1  BasicGame
2      SpriteSet
3          floor > Immovable img=oryx/grass autotiling=True hidden=
               True
4          cocoon > Immovable color=BLUE img=newset/cocoonb2
5          animal > physicstype=GridPhysics
6              avatar    > MovingAvatar img=oryx/angel1 frameRate=8
7              butterfly > RandomNPC speed=0.6 img=newset/butterfly1
                   cons=1 frameRate=5
8
9          wall > Immovable img=oryx/tree2
10
11     TerminationSet
12         SpriteCounter stype=butterfly win=True
13         SpriteCounter stype=cocoon    win=False
14
15     InteractionSet
16         animal    wall   > stepBack
17         butterfly avatar > killSprite scoreChange=2
18         butterfly cocoon > cloneSprite
19         cocoon butterfly > killSprite
20
21     LevelMapping
22         1 > floor butterfly
23         0 > floor cocoon
24         A > floor avatar
25         . > floor
26         w > floor wall
```

Figura 3.1: Código VGDL para o jogo Butteflies.

Para permitir que as descrições sejam concisas, uma ontologia subjacente define vários blocos de alto nível para os jogos, incluindo os tipos de físicas usadas, a dinâmica de movimento dos objetos e efeitos de interação durante colisões de objeto.

A VGDL descrita pode descrever uma grande variedade de vídeo games, incluindo versões aproximadas de jogos clássicos como *Pac-Man*, *Legend of Zelda* e *Mario Bros*. Informações em maiores detalhes sobre essa linguagem podem ser obtidas em (EBNER *et al.*, 2013).

3.3 java-VGDL

O java-VGDL é uma adaptação da versão inicial da VGDL (py-VGDL) para a linguagem Java, sendo o *framework* que apoia a competição GVG-AI. É fornecido em código aberto e oferece diversas funcionalidades que permitem a implementação e teste de controladores GVGP.

Este *framework* permite o carregamento de jogos descritos em VGDL e possui uma interface que permite a implementação de controladores que determinam as ações do jogador. Entretanto, a definição VGDL do jogo não é acessível diretamente ao controlador, sendo de responsabilidade do controlador determinar a natureza do jogo e a sequência de ações necessárias para alcançar a vitória através do *forward model*.

Neste *framework*, um controlador deve herdar de uma classe abstrata e implementar dois métodos: o construtor, que é chamado uma vez por jogo, e o método *act*, chamado a cada ciclo do jogo, que deve determinar a ação do controlador. O *framework* permite definir um tempo limite para a execução destes métodos, visto que vídeo games funcionam em tempo real e não podem esperar por muito tempo para a tomada de decisão do controlador (por padrão, na competição GVG-AI é adotado 1 segundo para o construtor e 40 milissegundos para cada execução do método *act*). Ambos os métodos recebem um objeto da classe *ElapsedCpuTimer* que fornece a hora atual e um objeto que representa o estado jogo.

3.3.1 Elementos do jogo

Cada *sprite* (elemento visual do jogo) pertence a uma categoria identificada por um ID, de forma que controladores possam identificar que dois elementos em posições distintas em um jogo são do mesmo tipo e cumprem a mesma função. Além disso, *sprites* também são agrupados em categorias que permitem inferir algumas de suas funções no jogo. São elas:

- NPCs: personagens não jogáveis (do inglês, *Non-Playable Characters* que pertencem ao jogo mas não podem ser controlados.
- *Resources*: recursos que podem ser coletados pelo jogador. Ao coletar um recurso,

alguma mudança no jogo ou nas propriedades do avatar será acionada.

- *Portals*: portais que permitem ao avatar mudar sua posição no jogo. Elementos dessa categoria podem ser usados como portas, buracos, escadas, etc.
- *Immovables*: elementos estáticos do jogo, normalmente utilizados como obstáculos, tais como paredes e árvores.
- *Movables*: elementos que podem ser movidos ou serem destruídos durante o jogo, mas que não são NPCs. Podem ser vistos como elementos do jogo que podem interagir com o avatar.

3.3.2 Estados

O objeto da classe *StateObservation* provê as seguintes informações para o controlador sobre o estado do jogo em determinado momento:

- Contagem de tempo atual, pontuação e estado do jogo (vitória, derrota ou em progresso)
- Lista de ações disponíveis para o avatar, podendo variar de jogo para jogo
- Lista de observações, cada elemento no jogo recebe um ID único, são agrupados por elementos do mesmo tipo, e pertencem à uma das categorias descritas anteriormente.
- Grade de observações, uma matriz representando o "mapa"do jogo, onde cada posição x, y da matriz contém as observações presentes naquela posição
- Histórico de eventos de colisão do avatar para aquele jogo

3.3.3 Ações

Um total de 6 ações possíveis para o jogador são disponibilizadas para serem usadas em jogos. Entretanto, um determinado jogo pode incluir apenas algumas dessas ações, de acordo com o estilo de jogabilidade a ser desenvolvido.

- *ACTION_UP*: move o avatar para cima

- *ACTION_DOWN*: move o avatar para baixo
- *ACTION_LEFT*: move o avatar para a esquerda
- *ACTION_RIGHT*: move o avatar para a direita
- *ACTION_USE*: ação específica de jogo (e.g. interagir com objetos, disparar um projétil, etc)
- *ACTION_NIL*: disponível para todos os jogos, não executa nenhuma ação

3.3.4 Forward model

O *forward model* permite ao controlador executar simulações a partir de determinado estado do jogo. Desta forma, o agente pode se guiar não somente por eventos que já aconteceram, mas também por eventos que acontecem durante simulações.

Durante a chamada de um método *act*, o controlador pode avançar o estado de um objeto *StateObservation* para um próximo estado executando-se uma simulação de uma determinada ação. Um objeto estado pode ser copiado e pode ser avançado tantos passos quanto desejado durante uma chamada, desde que esteja dentro do limite de tempo computacional disponível para o controlador decidir a ação que vai executar.

É importante observar que vários dos jogos fornecidos com o *framework* são de natureza estocástica, portanto, não existe garantia de que um estado obtido a partir de uma simulação será exatamente o mesmo estado resultante na execução de uma ação por parte do controlador durante o jogo. É responsabilidade do controlador, portanto, lidar com este fenômeno.

3.4 Modos de jogo

Desde a sua concepção em 2014, o *framework* da competição GVG-AI vem passando por diversas atualizações e melhorias, possibilitando maneiras diferentes de executar os jogos. Nesta seção serão descritas duas diferentes categorias a qual um jogo pode pertencer: uma relacionada ao número de jogadores e outra relacionada ao tipo de física de jogo.

3.4.1 Quantidade de jogadores

No que se refere à quantidade de jogadores, dois modos de jogo são disponibilizados. O modo *single player* (do inglês, jogador único), permite apenas um único jogador ou controlador em um mesmo jogo. Usualmente, este modo de jogo oferece como desafios derrotar todos os oponentes, capturar itens, sobreviver até o final, etc. Em ambiente GVGP, o principal desafio é descobrir qual destes objetivos o jogo propõe, fazendo assim com que o controlador possa ser direcionado à um estado de vitória.

Já o modo *multiplayer* (do inglês, multi jogador), permite que mais de um jogador ou controlador no mesmo jogo (atualmente, só existe suporte para 2 jogadores). Este modo pode ainda ser dividido em duas subcategorias: competitivos e cooperativos. No modo competitivo, a natureza do jogo pode ser entendida de maneira similar ao jogo *single player*, em que o segundo jogador seria um NPC hostil a ser derrotado. Por outro lado, jogos cooperativos podem oferecer desafios mais complexos, já que podem exigir coordenação nas ações por parte dos controladores.

Neste trabalho, tanto os modos de jogador único como os de multi jogador serão explorados para avaliar comparativamente os métodos propostos.

3.4.2 Tipo de física

Inicialmente, apenas jogos com física baseada em tabuleiro (do inglês, *grid physics*) eram suportados pelo *framework*. Como o próprio nome descreve, neste tipo de jogo um determinado elemento possui uma posição dada por valores discretos, mapeados em uma matriz que representa o mapa o jogo. Ao executar uma ação de movimento, a posição do avatar no próximo tempo do jogo será incrementado ou decrementado uma unidade em relação à posição anterior, funcionando de maneira similar a um jogo de tabuleiro.

Atualmente, o *framework* também suporta jogos de física contínua. Neste modo de jogo, elementos possuem posição com valores contínuos e podem estar sujeitos a efeitos de física como aceleração, gravidade, etc, de acordo com a descrição do jogo.

Como o modo de jogo de física contínua ainda se encontra em fase de testes, somente jogos com física de tabuleiro serão explorados neste trabalho.

4 Monte Carlo Tree Search

A busca em árvore de Monte Carlo (MCTS) é uma técnica de simulação de Monte Carlo para problemas de árvore de busca. Foi descrita pela primeira vez em 2006, em três diferentes trabalhos (COULOM; RÉMI, 2006; KOCSIS; LEVENT e SZEPESV; CSABA, 2006; CHASLOT *et al.*, 2006)).

Na execução do algoritmo MCTS, uma árvore de busca é construída através da realização de jogadas aleatórias utilizando um *forward model* e propagando o resultado obtido no estado final dessa jogada nos nós percorridos durante a busca. A cada iteração do algoritmo, uma estratégia de seleção define um nó a ser expandido e e processo descrito é repetido (FRYDENBERG *et al.*, 2015).

CHASLOT *et al.* (2008) discute a aplicação desta técnica em Inteligência Artificial para jogos, descrevendo os quatro passos básicos de um algoritmo baseado em MCTS. A Figura 4.1 apresenta um diagrama com uma representação da execução de um algoritmo MCTS em um controlador para jogos, onde o algoritmo usa uma árvore de possíveis estados futuros do jogo. A execução segue os seguintes passos:

- Seleção: a partir do nó raiz, uma estratégia de seleção de nós é aplicada até que se chegue a um nó folha.
- Expansão: a menos que o nó folha escolhido pela estratégia de seleção represente um estado onde o jogo terminou (com uma vitória, derrota ou empate), um novo nó é adicionado à árvore.
- Simulação: a partir do nó adicionado à árvore no passo anterior, uma simulação é executada descendo-se na árvore de acordo com uma estratégia de simulação (por exemplo, aleatória) até uma profundidade especificada.
- *Backpropagation*: uma recompensa associada ao estado final da simulação é propagada à todos os nós visitados durante a busca. O número de visitas de cada um desses nós também é atualizado.

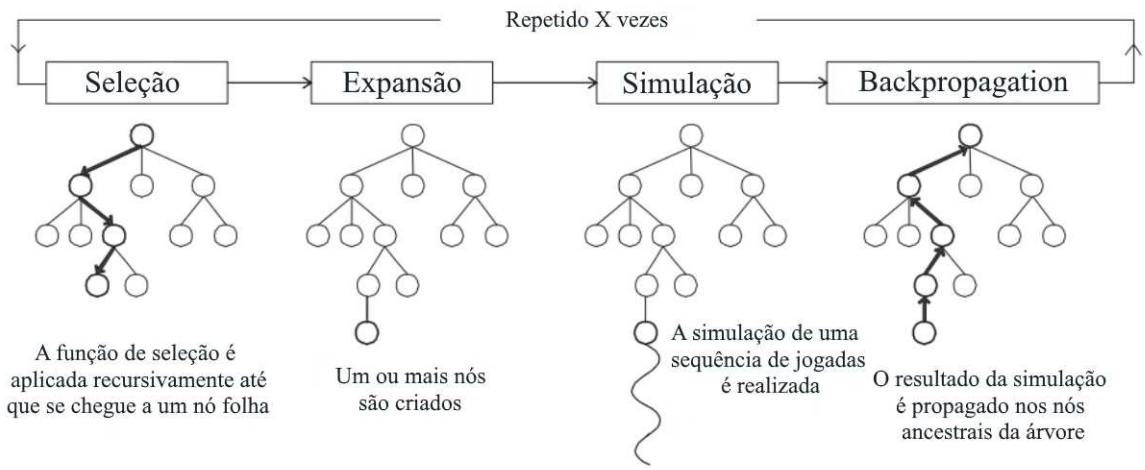

Figura 4.1: Representação do algoritmo MCTS (adaptado de CHASLOT *et al.* (2008)) .

Ao final das da execução do algoritmo (normalmente dado pelo esgotamento do tempo computacional disponível), a ação a ser finalmente tomada pelo controlador é obtida a partir da escolha do melhor nó filho do nó raiz. Diferentes políticas de recomendação podem ser adotadas para a escolha deste nó. Uma estratégia comum é escolher o nó com o maior número de visitas ou a maior média de pontuação. Embora as duas estratégias comumente resultem na mesma ação, existem situações em que essas duas políticas podem diferir (BROWNE *et al.*, 2012).

4.1 Estratégia de seleção

A cada iteração do algoritmo MCTS, um novo nó é escolhido para ser expandido e ter uma simulação executada a partir de um de seus filhos. A estratégia de seleção, portanto, é responsável por balancear a diversificação (*exploration*) e a intensificação (*exploitation*) da busca. Por um lado, deve-se buscar nós que possuem melhores recompensas e, por outro, os nós menos promissores ainda devem ser explorados devido à incerteza da avaliação.

Uma estratégia popularmente utilizada em GVGP é a *Upper Confidence bounds applied to Trees* (UCT), uma aplicação do *Upper Confidence Bounds 1* (UCB1) aplicada em árvores (KOCSIS; LEVENT e SZEPESV; CSABA, 2006), onde o valor de cada nó da árvore é dado por:

$$UCT = Q(s,a) + C\sqrt{\frac{\ln(N(s))}{N(s,a)}} \tag{4.1}$$

onde $Q(s,a)$ é a média de recompensas (pontuações) do nó obtido com a execução da ação a para o estado s, $N(s)$ é o número de visitas do nó pai, e $N(s,a)$ o número de visitas do nó filho. O primeiro termo da equação é o termo de intensificação, o segundo termo é o de diversificação e a constante C é um peso para equilibrar os dois termos. Se as recompensas forem normalizadas em $[0,1]$, um valor comum de C para jogos de um jogador é $\sqrt{2}$. Além disso, se existe uma ação a partir do estado s tal que $N(s,a) = 0$, esta ação deve ser escolhida antes da equação UCT ser aplicada (PEREZ *et al.*, 2015B).

4.2 Estratégia de simulação

A estratégia de simulação é responsável por decidir quais ações serão executadas em sequência durante uma simulação. Em controladores para jogos específicos, a estratégia de simulação pode melhorar o desempenho do controlador de maneira significativa através do uso de conhecimento de domínio (BOUZY, 2005; MUNOS *et al.*, 2006).

Desenvolver uma boa estratégia de simulação é normalmente uma tarefa de equilibrar dois diferentes aspectos da simulação. Um é o equilíbrio entre conhecimento e busca. Adicionar conhecimento de domínio à simulação faz com que melhores jogadas sejam exploradas, tornando as jogadas mais confiáveis. Entretanto, se este conhecimento de domínio for caro computacionalmente, o número de simulações será reduzido, fazendo com que a árvore seja menos explorada e consequentemente reduzindo o desempenho do controlador. Por outro lado, há o equilíbrio entre diversificação e intensificação. Se a estratégia de simulação for muito aleatória, a diversificação será muito alta e muitas jogadas ruins serão exploradas. Entretanto, se as simulações forem enviesadas em um espaço de busca selecionado, a intensificação será muito alta e caminhos potencialmente promissores deixarão de ser explorados. Em ambos os casos, o desempenho do controlador também é prejudicado. (CHASLOT, 2010).

No caso de controladores GVGP, em que se desconhece previamente as regras do jogo, em geral, torna-se difícil (e indesejável) adicionar conhecimento de domínio à estraté-

gia de simulação. Embora algumas abordagens para o problema já tenham sido propostas, como direcionar a simulação em direção à outros elementos do jogo que produzam um aumento de pontuação (PEREZ; SAMOTHRAKIS; LUCAS, 2014), uma estratégia comum e que apresenta bons resultados é uma estratégia de simulação randômica.

4.3 Política de recomendação

A política de recomendação é a responsável por, ao final das simulações, decidir qual das ações será executada pelo controlador (qual dos nós filhos do nó raiz será escolhido). CHASLOT (2010) expõe quatro diferentes técnicas comumente utilizadas como políticas de recomendação em controladores que utilizam o algoritmo de MCTS:

- *Max child*: o nó com o maior valor é escolhido.
- *Robust child*: o nó com o maior número de visitas é escolhido.
- *Robust-max child*: o nó com ambos maior valor e número de visitas é escolhido. Se não há nenhum *Robust-max child*, novas simulações são executadas até que se obtenha um (COULOM; RÉMI, 2006).
- *Secure child*: o nó que maximiza um limite inferior de confiança, por exemplo, que maximiza $v+\frac{A}{n}$, onde v é o valor do nó, n é o número de visitas e A é uma constante.

4.4 *Open loop vs. Closed loop*

PEREZ *et al.* (2015B) discute sobre duas formas de implementar a busca no que se refere ao armazenamento de estados na árvore de busca. Em jogos determinísticos, os estados armazenados em uma árvore de determinado jogo são reproduções exatas do estado alcançado naquele nó dado que a sequência de ações representada pela sequência de nós seja executada. Assim, quando executado o *forward model* para uma ação a a partir de um estado estado s, o estado resultante s' é armazenado junto com o nó, e nas próxima vezes em que o algoritmo percorre a árvore passando por este nó, o mesmo estado s' armazenado é utilizado para prosseguir com a busca. Esta forma de implementação do algoritmo é conhecida como *Closed loop* MCTS.

Entretanto, em jogos estocásticos, não podemos garantir que os estados obtidos durante as simulações são representações fiéis dos estados do jogo dado que determinada sequência de ações é executada. Pode-se assumir que o estado resultante da simulação é obtido a partir de uma distribuição de probabilidades desconhecida. Neste sentido, torna-se indesejável armazenar estados em nós pois o estado gerado quando o nó foi adicionado à árvore pode não ser o estado *s'* com maior probabilidade de ser alcançado após executar a ação *a* para o estado *s*. Idealmente, o algoritmo deve simular cada uma das ações disponíveis um número suficiente de vezes para obter todos os estados possíveis a partir de determinado estado. Porém, esta estratégia mostra-se inviável para este cenário, já que o número de nós cresceria exponencialmente.

Uma maneira de mitigar este problema é, durante a execução do algoritmo, não armazenar os estados na árvore; apenas as estatísticas. A cada vez que o algoritmo percorre a árvore, o *forward model* é reexecutado para cada ação da cadeia de nós. Embora esta estratégia aumente o custo computacional, as estatísticas obtidas são uma representação do conjunto de estados possíveis de uma sequência de ações a partir de determinado estado, e não de uma sequência de pares (*s*, *a*). Essa forma de implementação do algoritmo é conhecida como *Open loop* MCTS e mostra-se mais apropriada para controladores GVGP já que podem lidar tanto com jogos determinísticos quanto estocásticos.

4.5 Características do algoritmo MCTS

FRYDENBERG *et al.* (2015) discorre sobre as características da busca em árvores Monte Carlo que permitem seu bom desempenho em GVGP:

- *Anytime*: a busca do algoritmo pode ser interrompida a qualquer instante, retornando a melhor ação encontrada até aquele momento.
- *Non-heuristic*: a busca não depende de conhecimento de domínio, basta um conjunto de operadores legais (ações possíveis a partir de determinado estado) para que a busca seja realizada. Essa é uma característica importante do algoritmo já que controladores GVGP não possuem conhecimento prévio sobre as regras do jogo.
- *Asymmetric*: ao invés de mapear o espaço de busca inteiro, o algoritmo de MCTS

concentra seus esforços em áreas promissoras anteriormente, construindo uma árvore assimétrica. Essa é uma característica crucial para aplicações em tempo real.

4.6 Desvantagens do algoritmo MCTS

Embora a busca em árvores Monte Carlo tenha se tornado um algoritmo popular em GVGP, algumas de suas características, em sua versão original, podem limitar seu desempenho (PEREZ; SAMOTHRAKIS; LUCAS, 2014):

- Profundidade da simulação: a grande maioria das simulações não encontrará um estado onde o jogo termina, impedindo o algoritmo de encontrar estados de vitória. Entretanto, dada a natureza de tempo real do problema e a impossibilidade de usar conhecimento de domínio para guiar a simulação, esta dificuldade não pode ser evitada.

- Simulações não informativas: se em determinado estado do jogo, todos os NPCs e objetos estão além do limite de profundidade de simulação do algoritmo, as simulações não retornarão recompensas. Não haverá incentivo para interagir com esses elementos e o agente passará a se mover aleatoriamente.

- Simulações redundantes: com uma estratégia de simulação randômica, sequências de ações que se anulam em determinados casos podem ser executadas (e.g. um movimento para a direita, seguido por um movimento para a esquerda). É importante notar que, em determinados jogos ou situações, tais sequências de ações podem ser interessantes de serem executadas.

- Uso de eventos passados: informações de eventos obtidos em momentos anteriores do jogo (e.g. uma colisão com um NPC que retorna um aumento na pontuação) não são utilizados para guiar o agente novamente a eventos similares.

5 Melhorias propostas

Durante a elaboração deste trabalho, algumas modificações para o algoritmo de busca em árvores Monte Carlo foram elaboradas. Este capítulo irá detalhar estas modificações.

5.1 Desencorajar ações redundantes

Uma estratégia de simulação comum para o MCTS é a randômica. Como em GVGP as informações de regras dos jogos não são passadas diretamente aos controladores, desenvolver estratégias para guiar a simulação torna-se um desafio.

Estratégias específicas (como por exemplo, guiar o controlador até o elemento mais próximo) podem funcionar muito bem para determinados jogos, mas não serão adequadas à outros. Além disto, essas estratégias podem aumentar o custo computacional da simulação e, consequentemente, diminuir o número de iterações da busca, reduzindo o desempenho geral do controlador.

Por outro lado, conforme observado na seção de desvantagens do MCTS do Capítulo 4, em uma estratégia de simulação randômica, sequências de ações redundantes são frequentemente executadas. Portanto, torna-se interessante estabelecer uma política para diminuir a ocorrência de ações redundantes, porém, sem eliminá-las por completo, já que podem existir situações em que tais ações são interessantes de serem executadas.

Primeiro, deve-se definir o que são ações redundantes. Uma sequência de ações-estado (a_1, s_1), (a_2, s_2) é redundante se e somente se a execução de ambas as ações a partir de um estado s_0 não resultarem em uma mudança de estado do avatar. Uma mudança de estado do avatar pode ser dada por:

- Posição: se a ação a1 causou no avatar uma mudança de posição p_0 para a posição p_1, a ação a_2 não deve causar uma nova mudança de posição para p_0
- Orientação: se a ação a_1 causou uma mudança de orientação no avatar, a ação a_2 não deve causar uma nova mudança de orientação.

- Objetos criados pelo avatar: se a ação a_1 produziu um novo objeto, a ação a2 não deve ser nula (deve causar uma nova mudança de posição, orientação, ou criação/-destruição de objeto).

É importante observar que essa definição inclui apenas movimentação e ação do avatar que não envolvam outros elementos do jogo. Toda a ação que gere uma colisão com outro elemento do jogo é desconsiderada, já que a identificação de interações redundantes é mais complexa e fora do escopo deste trabalho.

Inicialmente, define-se uma distribuição uniforme de probabilidades todas as transições de ações disponíveis como:

$$P(a, s) = \frac{1}{n} \tag{5.1}$$

onde *P(a, s)* é a probabilidade de escolha da ação a a partir do estado s e n é o número de ações disponíveis. Dado que a ação a foi escolhida para o estado s, e existe uma ação a' tal que a' é redundante em relação à a, a probabilidade de escolha da ação a' para o estado s' é dada por:

$$P(a', s') = \frac{1}{n} * \frac{p}{r} \tag{5.2}$$

onde p é um termo de penalização e r é o número total de ações redundantes para o estado s', equilibrando o valor da penalização, caso haja mais de uma ação redundante. Neste trabalho, o valor de p foi definido como 0.5 de maneira empírica. O valor penalizado das ações redundantes é redistribuído entre as demais ações não-redundantes. Nota-se que, caso ações redundantes não tenham sido identificadas, a distribuição uniforme inicial se mantém.

A identificação de ações redundantes se dá no momento em que o controlador é inicializado e pode ser descrita pelo algoritmo da figura 5.1.

Caso a posição inicial do avatar seja cercada por obstáculos, não será possível identificar todas as possíveis ações redundantes do jogo. Pode-se estabelecer uma estratégia para, caso a posição inicial do avatar possua algum obstáculo ao redor, mudar a posição do mesmo até que se encontre uma área onde seja possível fazer a devida verificação de ações redundantes. Entretanto, essa tarefa pode se tornar complexa. Independentemente da estratégia utilizada, não é possível fornecer nenhuma garantia sobre a possibilidade

```
1  Inicio
2       a  <- acoes disponiveis
3       s0 <- estado atual
4       Para toda acao a1 em a
5            s1 <- (s0, a1)
6
7            // ocorreu colisao, ignora
8            Se s0.eventos != s1.eventos entao
9                 continua
10
11           Para toda acao a2 em a
12                s2 <- (s1, a2)
13
14                // identificada uma acao de movimento nula
15                Se s0.posicao != s1.posicao e s2.posicao == s0.
                     posicao entao
16                  a1.redundantes = a1.redundantes + a2
17                  continua
18                Fim se
19
20                // identificada acao de disparo seguida de acao nula
21                Se s0.objetos != s1.objetos entao
22                    Se s1.posicao == s2.posicao
23                      e s1.orientacao == s2.orientacao
24                      e s1.objetos == s2.objetos entao
25                         a1.redundantes = a1.redundantes + a2
26                    continua
27                Fim se
28
29           Fim para
30
31      Fim para
32 Fim
```

Figura 5.1: Pseudo-código para o algoritmo de identificação de ações redundantes.

de obter todas as ações redundantes para uma determinada partida. Na prática, a solução proposta conseguiu identificar ações redundantes para até 60% das ações dos jogos testados.

5.2 Evitar ações de derrota

Em situações onde várias sequências de ações resultam em uma derrota no jogo, a escolha da ação a ser executada pelo controlador no algoritmo MCTS muitas vezes se reduz a uma escolha randômica. Isso acontece pelo fato das simulações, dado sua característica aleatória, falharem em encontrar uma sequência de ações que não resulte em uma derrota. Dessa forma, todas as simulações retornarão uma recompensa negativa e a escolha será

dada pelo nó com maior número de visitas. Esse problema pode ser corrigido com uma execução de simulações suficientes pelo algoritmo, de forma a encontrar ao menos uma simulação (sequência de ações) que não leve a uma derrota, executando a primeira ação pertencente a tal sequência. Entretanto, dado a natureza do problema de GVGP, essa solução não se torna viável.

SOEMERS *et al.* (2016) propõe uma estratégia para contornar este problema durante as simulações. Ao descer na árvore durante uma simulação, caso o nó atual na busca represente uma derrota, retornar ao nó pai e continuar a simulação a partir de outro filho (a partir de outra ação), até que se obtenha uma sequência de ações que não tenha resultado em uma derrota. Segundo experimentos dos autores, essa é uma estratégia que funciona bem para determinados jogos, entretanto, observa-se que o número de simulações em comparação ao algoritmo original pode ser reduzido até pela metade, prejudicando o desempenho nos demais jogos.

Neste trabalho, propõe-se uma estratégia mais simples. Considerando-se uma implementação *open loop*, durante as etapas de seleção e expansão do algoritmo, sempre que um estado de derrota for identificado naquele nó, marcá-lo como um nó de derrota. Ao final das simulações, escolher um nó para ser executado somente dado que ele nunca foi marcado como nó de derrota, a não ser que seja a única opção. Essa estratégia garante que, dado um perigo iminente de derrota, caso exista uma ação que possibilite a continuação do jogo (evitar ou adiar a derrota), esta ação é escolhida.

6 Experimentos e análise

Os métodos propostos serão aplicados tanto em jogos de jogador único como em jogos de dois jogadores. Para jogos de dois jogadores, também serão utilizados os dados da competição 2-P GVG-AI divulgados na *IEEE Conference on Evolutionary Computation 2017* (CEC 2017)[2].

Antes de verificar o desempenho das modificações propostas, a busca em árvores Monte Carlo em sua forma *vanilla* (sem quaisquer modificações) foi testada para se obter uma base de comparação.

Para garantir maior diversidade aos jogos a serem utilizados no conjunto de testes, utilizou-se o conjunto de jogos descrito em GAINA *et al.* (2017). No referido trabalho, duas diferentes classificações para os jogos utilizados na GVG-AI propostas na literatura serviram de base para a composição de um conjunto diverso de jogos. O primeiro método de classificação, proposto em NELSON (2016), classifica os jogos de acordo com sua quantidade de vitórias para o MCTS *vanilla*. Já o segundo, proposto em BONTRAGER *et al.* (2016), classifica os jogos em 4 categorias de acordo com as características do jogo. A lista de jogos selecionados é exibida na Tabela 6.1.

Nome	Tipo	Nome	Tipo
Aliens	E	Bait	D
Butterflies	E	Camel Race	D
Chopper	E	Chase	D
Crossfire	E	Escape	D
Digdug	E	Hungry Birds	D
Infection	E	Lemmings	D
Intersection	E	Missile Command	D
Roguelike	E	Modality	D
Seaquest	E	Plaque Attack	D
Survive Zombies	E	Wait for breakfast	D

Tabela 6.1: Jogos de jogador único selecionados para fazer parte do conjunto de testes. Legenda: D - Determinístico, E - Estocástico

As modificações propostas neste trabalho foram executadas para os 20 jogos, em todos os 5 níveis, 20 vezes cada, resultando em 100 partidas para cada jogo por cada

[2]http://www.cec2017.org/

controlador. Para fins de comparação, a quantidade de vitórias e a pontuação média para cada jogo serão avaliados. Nota-se que cada jogo possui um sistema de pontuação próprio, não sendo interessante calcular a média de pontuação geral em todos os jogos, e sim analisar a média jogo por jogo.

O limite computacional de cada execução seguirá o padrão utilizado pela competição GVG-AI, 40 milissegundos de tempo computacional para o retorno de uma ação a ser executada. Caso o tempo computacional gasto fique entre 40 e 50 milissegundos, a ação *ACTION_NIL* (ação nula) é executada. Caso o controlador ultrapasse os 50 milissegundos, o mesmo é desclassificado imediatamente e recebe uma pontuação negativa. Neste trabalho, partidas em que o controlador for desclassificado, caso ocorram, não serão contabilizadas nas estatísticas do controlador. É importante observar que os resultados podem variar dependendo do ambiente computacional em que for executado. Devido à limitação de tempo computacional, as mesmas técnicas descritas neste trabalho aplicadas em um ambiente com maior capacidade de processamento provavelmente serão capazes de retornar resultados melhores do que os apresentados neste trabalho. Portanto, torna-se mais interessante observar o desempenho relativo entre os controladores do que os valores absolutos de desempenho de um único controlador.

As modificações propostas, desencorajar ações redundantes e evitar nós de derrota, serão referidas nos experimentos como MCTS+MR e MCTS+ND, respectivamente. O controlador que implemente as duas modificações será referido como MCTS+MR+ND.

6.1 Resultados

6.1.1 Jogador único

O resultado das execuções são apresentadas na Tabela 6.2. Inicialmente, pode-se perceber que nenhuma das estratégias é dominante em relação às outras. De fato, embora o MCTS *vanilla* tenha obtido a maior porcentagem de vitórias em apenas 2 jogos, para a maioria dos jogos restantes a diferença para o MCTS+MR ou para o MCTS+ND é mínima, resultante da aleatoriedade das execuções.

Outro ponto importante a ser observado é que alguns jogos (Escape, Lemmings,

Roguelike, Dig Dug) resultaram em 0% de vitórias entre todos os controladores, exceto pelos 1% de vitórias resultantes do MCTS+ND para o Dig Dug. Este resultado pode ser explicado pelo fato destes jogos exigirem uma longa sequência de ações específicas sem que nenhuma recompensa (pontuação) seja fornecida ao agente, tornando-se inviável para uma estratégia com MCTS que usa como função de avaliação a pontuação do jogo. Destes jogos, dois merecem ser detalhados individualmente, o que será feito a seguir.

Nome	MCTS	MCTS+MR	MCTS+ND	MCTS+MR+ND
Aliens	98% (69.10 +/- 14.92)	98% (69.01 +/- 15.18)	97% (68.99 +/- 15.12)	**100% (68.73 +/- 15.12)**
Bait	8% (2.59 +/- 2.75)	**12% (2.65 +/- 2.50)**	11% (4.77 +/- 6.64)	7% (2.33 +/- 2.53)
Butterflies	97% (31.28 +/- 16.34)	99% (30.32 +/- 15.35)	96% (29.46 +/- 15.88)	**100% (29.69 +/- 14.49)**
Camel Race	**6% (-0.74 +/- 0.56)**	3% (-0.77 +/- 0.49)	3% (-0.77 +/- 0.49)	3% (-0.76 +/- 0.51)
Chase	7% (3.21 +/- 2.19)	8% (3.26 +/- 1.75)	**17% (4.64 +/- 2.66)**	2% (2.94 +/- 1.85)
Chopper	17% (-1.94 +/- 7.50)	22% (-0.86 +/- 6.73)	**29% (0.35 +/- 7.51)**	8% (-4.02 +/- 7.67)
Crossfire	2 % (0.07 +/- 0.72)	**4% (0.18 +/- 0.99)**	3 % (0.15 +/- 0.85)	0% (0.00 +/- 0.00)
Digdug	0 % (11.38 +/- 9.22)	0 % (12.43 +/- 8.87)	**1 % (11.87 +/- 9.98)**	0% (12.19 +/- 9.81)
Escape	0 % (0.00 +/- 0.00)	0% (0.00 +/- 0.00)	0 % (0.00 +/- 0.00)	0% (0.00 +/- 0.00)
Hungry Birds	5 % (5.80 +/- 22.32)	6% (6.40 +/- 23.98)	4 % (4.80 +/- 20.22)	**10% (12.40 +/- 30.70)**
Infection	95% (14.47 +/- 8.62)	96% (13.74 +/- 7.61)	**99% (14.52 +/- 8.59)**	97% (14.86 +/- 9.76)
Intersection	95% (14.47 +/- 8.62)	**100% (1.00 +/- 0.00)**	**100% (1.00 +/- 0.00)**	**100% (1.00 +/- 0.00)**
Lemmings	0 % (-3.91 +/- 2.91)	0 % (-4.09 +/- 3.55)	**0 % (-3.29 +/- 2.86)**	0% (-3.83 +/- 3.31)
Missile Command	61% (4.41 +/- 5.16)	**65% (4.64 +/- 4.93)**	59% (4.20 +/- 4.76)	64% (4.04 +/- 4.20)
Modality	27% (0.27 +/- 0.44)	**30% (0.30 +/- 0.46)**	29% (0.29 +/- 0.45)	26% (0.26 +/- 0.44)
Plaque Attack	90% (46.36 +/- 18.59)	85% (44.76 +/- 21.22)	95% (48.19 +/- 17.65)	**96% (49.52 +/- 18.65)**
Roguelike	0 % (3.82 +/- 4.66)	0 % (3.34 +/- 4.21)	0 % (4.40 +/- 6.32)	**0% (4.44 +/- 4.93)**
Seaquest	58% (1887 +/- 1996)	62 % (2136 +/- 2267)	34 % (1648 +/- 2082)	**76% (2329 +/- 2236)**
Survive Zombies	42% (2.95 +/- 3.70)	40% (2.80 +/- 3.57)	**45% (3.17 +/- 3.67)**	44% (3.14 +/- 3.67)
Wait for breakfast	9 % (0.09 +/- 0.29)	10% (0.10 +/- 0.30)	**52% (0.54 +/- 0.50)**	4% (0.04 +/- 0.20)

Tabela 6.2: Resultado da execução dos jogos. O controlador com a maior porcentagem de vitórias para cada jogo está marcada em negrito. O valor entre parênteses indica a pontuação média e o desvio padrão

Escape é um jogo determinístico que propõe um quebra cabeça simples, onde o personagem deve empurrar blocos presentes no caminho até chegar ao objetivo. Entretanto, existem buracos muito próximos aos blocos, que levam à uma derrota caso o jogador passe por cima de um deles. Todas as simulações que passem por esses buracos retornarão uma recompensa negativa no MCTS e, portanto, o avatar nunca irá explorar os caminhos próximos aos buracos, onde os blocos devem ser empurrados. Nota-se que, como este é um jogo determinístico sem nenhum tipo de oponente ou componente dinâmico de jogo, uma simples busca em largura, executada continuamente por vários passos do jogo, eventualmente retornaria uma sequência de ações que resolveria o quebra cabeça. Entretanto, como o foco deste trabalho é no desempenho do MCTS, optou-se por não utilizar um controlador com uma implementação híbrida.

Já no jogo Lemmings, diversos NPCs surgem de um determinado local do mapa,

e seguem através de múltiplos caminhos até um determinado ponto objetivo. Porém, os caminhos que eles precisam percorrer são bloqueados por paredes. Neste jogo, o jogador deve destruir as paredes que ficam no caminho dos NPCs e permitir que eles cheguem até seu objetivo. Entretanto, o jogador tem a pontuação reduzida ao destruir alguma dessas paredes. Ou seja, o agente deve executar ações que resultam na perda de pontuação a fim de obter a vitória, o que aumenta a dificuldade do problema, já que a perda de pontuação passa a indicar um progresso positivo durante a partida.

O controlador MCTS+ND apresentou o melhor resultado para os jogos Chase, Chopper, Dig Dug, Infection, Intersection, Plague Attack, Survive Zombies e Wait for Breakfast. Torna-se necessário analisar as características dos jogos para identificar se a melhora de desempenho se deu devido às modificações se ajustarem bem a essas características ou devido a aleatoriedade das execuções.

Os resultados mais simples de serem analisados são dos jogos Chase, Chopper, Plague Attack e Survive Zombies. Todos estes jogos possuem, como uma de suas características, fugir de ameaças como oponentes ou projéteis que, entrando em contato com a avatar, levam à perda do jogo. Além disso, essas ameaças podem perseguir o jogador ou vir de múltiplas direções, dificultando ao MCTS *vanilla* encontrar a melhor ação em uma situação de perigo imediato. No caso destes jogos, a estratégia proposta pode trazer resultados melhores. Já para os jogos Dig Dug, Infection, Intersection e Wait for Breakfast, a análise passa a ser mais complicada, já que estes jogos não possuem características que possam ser beneficiadas pela estratégia. Pode-se dizer que a melhoria no desempenho nestes casos ocorreu devido à aleatoriedade das execuções. É importante notar que a modificação proposta no MCTS+ND não oferece uma estratégia vencedora para nenhum desses jogos: ela apenas garante a escolha da melhor ação em algumas situações que, no MCTS *vanilla*, poderiam ser aleatórias e levar a uma derrota, com uma sobrecarga mínima sobre a execução do algoritmo.

Já o controlador MCTS+MR apresentou o melhor resultado para os jogos Bait, Butterflies, Crossfire, Hungry Birds, Intersection, Missile Command, Modality e Seaquest. Diferente do MCTS+ND, que oferece uma estratégia que sempre será executada em determinadas situações, podendo garantir maior duração ou possivelmente uma vitória no

jogo, o MCTS+MR visa apenas executar simulações mais eficazes, o que pode ou não melhorar os resultados de deterimado jogo. De fato, todos estes jogos oferecem como um de seus principais desafios a movimentação pelo mapa do jogo, que poderiam ter seu desempenho aprimorado com menos ações redundantes na simulação. Entretanto, a diferença entre a porcentagem de vitórias deste controlador e do MCTS *vanilla* não é suficiente para fazer qualquer afirmação. Neste caso, a qualidade das simulações e o impacto sobre a quantidade total de simulações executadas serão analisados.

O desempenho do algoritmo de MCTS é diretamente ligado a quantidade de simulações executadas. Quanto mais simulações executadas, mais certo se está sobre a validade dos estados avaliados. Ao propor modificações para o MCTS, é importante garantir que o impacto na quantidade de simulações não seja grande. A Tabela 6.3 mostra a quantidade média de simulações executadas nos jogos para cada um dos controladores. É possível observar que não houve variação significativa na média de simulações por jogo. Isto significa que, mesmo que que as modificações propostas não garantam um pequeno aumento no número de vitórias do controlador, também não irão impactar negativamente o processo de busca do algoritmo para jogos em que essas modificações não sejam relevantes.

Game	MCTS	MCTS+MR	MCTS+ND	MCTS+MR+ND
Aliens	78.12	78.56	79.8	77.68
Bait	123.08	126.08	132.52	141.52
Butterflies	53.33	54.5	54.57	53.1
Camel Race	50.7	51.47	51.13	51.1
Chase	49.43	50.9	50.7	50.07
Chopper	33.93	34.3	34.53	35.13
Crossfire	30.87	31.27	31.07	31.07
Digdug	22.4	22.23	22.37	21.9
Escape	60.9	62.07	61.27	61.43
Hungry Birds	91.6	92.23	92.37	91.83
Infection	44.07	44.4	44.13	44.33
Intersection	60.97	61.2	60.87	60.77
Lemmings	41.23	41.1	41.27	42.17
Missile Command	92.1	91.8	92.0	90.77
Modality	260.23	258.7	257.3	254.17
Plaque Attack	30.3	30.0	29.77	29.47
Roguelike	28.03	28.07	27.97	27.9
Seaquest	75.72	72.0	72.8	68.44
Survive |ombies	42.33	41.73	42.7	43.9
Wait For Breakfast	147.43	145.7	147.73	144.53

Tabela 6.3: Quantidade média de simulações a cada passo de tempo para cada jogo.

Game	MCTS *vanilla*	MCTS+MR
Aliens	1.59	0.89
Bait	0.65	0.43
Butterflies	1.49	0.78
Camel Race	0.84	0.74
Chase	1.1	0.66
Chopper	1.1	0.54
Crossfire	1.16	0.64
Digdug	0.24	0.21
Escape	0.71	0.6
Hungrybirds	0.69	0.42
Infection	0.27	0.12
Intersection	1.42	0.78
Lemmings	0.27	0.12
Missile Command	0.24	0.22
Modality	0.75	0.53
Plaque Attack	0.28	0.25
Roguelike	-	-
Seaquest	0.26	0.24
Survive Zombies	0.77	0.48
Wait for Breakfast	0.86	0.67

Tabela 6.4: Quantidade média de ações redundantes por simulação.

Já a Tabela 6.4 mostra a ocorrência de ações redundantes entre o MCTS *vanilla* e o MCTS com a modificação proposta. Nota-se que os jogos que utilizam características de orientação em sua maioria não foram afetados (em especial para o jogo Roguelike, em que nenhuma ação redundante foi identificada em nenhum dos níveis do jogo), mas para os demais a modificação houve uma significativa queda na média de ações redundantes por simulação.

Estes números mostram que o método proposto foi eficiente em reduzir a ocorrência de ações redundantes, em casos em que essas ações conseguiram ser identificadas. Este resultado sugere um maior refinamento do método proposto, além de sua aplicação em outros algoritmos além do MCTS, que também poderiam se beneficiar com uma redução da ocorrência de ações redundantes.

6.1.2 Dois jogadores

Para expandir a avaliação das técnicas propostas neste trabalho, um controlador implementando as duas modificações propostas no Capítulo 5 foi submetido para a competição

2-P GVG-AI 2017. Neste modo de jogo, dois controladores são carregados em uma mesma partida e precisam colaborar ou competir (de acordo com as regras do jogo) para obter a vitória.

Primeiramente, o controlador foi avaliado em um dos conjuntos de teste disponibilizados pela competição. A lista de jogos e a porcentagem de vitórias para cada controlador se encontram na Tabela 6.5. No total, três controladores foram utilizados na execução deste experimento: o MCTS+MR+ND, MCTS *vanilla*, para fazer uma comparação direta com o controlador que implementa as modificações propostas para o algoritmo, e o *Sample Random Controller* (retorna uma ação aleatória a cada passo do jogo), adicionado ao conjunto para garantir maior diversidade aos resultados.

Tipo	Jogo	*sampleRandom*	*sampleOLMCTS*	*ehauckdo*
Cl	Akkaarrh	0% (1.23 +/- 4.02)	2% (11.60 +/- 14.77)	2% (14.62 +/- 14.90)
Cp	Asteroids	2% (2.24 +/- 4.85)	68% (43.09 +/- 51.74)	**73% (42.09 +/- 54.97)**
Cp	Capture Flag	2% (-4.53 +/- 3.82)	**81% (9.65 +/- 8.68)**	68% (7.69 +/- 8.97)
Cl	Compete Sokoban	0% (0.05 +/- 0.27)	0% (1.09 +/- 1.50)	0% (0.99 +/- 1.51)
Cp	Cops N Robbers	0% (0.53 +/- 0.97)	43% (2.61 +/- 2.14)	**53% (2.96 +/- 2.05)**
Cp	Gotcha	50% (209.25 +/- 438.80)	50% (457.80 +/- 701.89)	50% (245.22 +/- 498.64)
Cp	Klax	0% (0.33 +/- 0.82)	**77% (12.78 +/- 8.21)**	73% (13.25 +/- 8.35)
Cp	Samaritan	40% (0.40 +/- 0.49)	52% (0.54 +/- 0.50)	**55% (0.55 +/- 0.50)**
Cp	Steeple Chase	1% (87.51 +/- 282.61)	**6% (173.40 +/- 378.58)**	3% (141.06 +/- 348.11)
Cp	Tron	12% (0.14 +/- 0.34)	62% (0.64 +/- 0.48)	**75% (0.76 +/- 0.43)**

Tabela 6.5: Comparação de resultados entre o MCTS vanilla e o controlador submetido. Legenda: Cp - Competitivo, Cl - Colaborativo

De maneira similar ao que ocorreu nos experimentos de jogos de jogador único, todos os controladores testados obtiveram 0% de vitórias para o jogo Compete Sokoban. Este jogo requer uma colaboração complexa entre os jogadores, no sentido de mover blocos e resolver um quebra cabeça. O controlador randômico também obteve 0% de vitórias para alguns jogos. Porém, este resultado era esperado devido à sua natureza.

Pode-se observar que o controlador com as modificações propostas obteve um pequeno aumento no número de vitórias para os jogos Asteroids, Cops N Robbers, Samaritan e Tron. Todos estes jogos se beneficiam de alguma maneira de ambas as estratégias propostas. Em especial, para o jogo Tron, o método de nós de derrota é capaz de fazer com que o controlador evite diversas situações óbvias de derrota, que o MCTS *vanilla*, conforme descrito no Capítulo 5, não é capaz de evitar. A quantidade de vitórias para os dois controladores evidenciam essa diferença.

Por outro lado, o MCTS *vanilla* continuou apresentando resultados melhores para alguns jogos (Capture Flag, Klax, Steeple Chase), mesmo que por uma margem pequena. Um raciocínio semelhante ao utilizado nos jogos de jogador único pode ser aplicado para estes resultados.

A Figura 6.1 mostra a classificação final da competição 2-P GVG-AI 2017[3]. Nota-se que, embora não se tenha informações detalhadas sobre os métodos implementados pelos outros controladores, o MCTS *vanilla* (nomeado *sampleOLMCTS* na competição) foi incluído pelos organizadores em para fins de comparação com os métodos propostos pelos participantes. O foco da análise será na comparação entre o MCTS *vanilla* e o controlador submetido na competição.

Observe que a competição usa o sistema de pontuação da Fórmula 1 na montagem da classificação, além de um sistema de pontuação que leva em consideração como um controlador se saiu em comparação a todos os outros oponentes em partidas executadas dois a dois (*round robin*). Portanto, embora se tenha uma grande diferença de colocação entre o controlador submetido e o MCTS *vanilla*, a diferença no que tange a porcentagem de vitórias e pontuação não é tão acentuada. Mais detalhes sobre o sistema de ranqueamento utilizado pode ser encontrado em GAINA *et al.* (2016).

Até a data de publicação deste trabalho, os jogos utilizados no conjunto de teste ainda não foram divulgados. Portanto, não é possível fazer uma avaliação jogo a jogo como foi feito para os testes de jogador único.

A Tabela 6.6 apresenta a porcentagem de vitórias para o MCTS *vanilla* e para o controlador submetido para a competição. O controlador submetido implementa os dois métodos descritos no Capítulo 5 deste trabalho.

	Game 1	Game 2	Game 3	Game 4	Game 5	Game 6	Game7	Game 8	Game 9	Game 10
sampleOLMCTS	**1%**	45%	49%	**51%**	34%	34%	54%	51%	**19%**	54%
ehauckdo	0%	**62%**	**50%**	50%	**37%**	**43%**	**60%**	**65%**	**19%**	**81%**

Tabela 6.6: Comparação de resultados entre o MCTS vanilla e o controlador submetido.

A primeira observação a ser feita é que os resultados parecem consistentes com os apresentados nos experimentos anteriores: a diferença de desempenho entre os dois métodos é pequena, com o método proposto levando uma pequena vantagem para alguns

[3]disponível em http://www.gvgai.net/gvg_rankings_2p.php?rg=2006

Rank	Username	Country	Points	Avg. Glicko Score
1	ToVo2	Slovenia	161	1183.41
2	ehauckdo	Brazil	111	1133.4
3	not2048	United Kingdom	111	1071.82
4	essex_acwebb	United Kingdom	110	1120.63
5	Number27	Germany	84	1124.69
6	sampleRS Sample	United Kingdom	78	1108.92
7	adrienctx	Belgium	66	1086.89
8	DreamTeam	United Kingdom	60	1096.42
9	PrettyTeam	United Kingdom	56	1023.9
10	sampleOLMCTS Sample	United Kingdom	55	1084.8
11	sampleRHEA Sample	United Kingdom	43	1059.52
12	SpaceJohn_Team	United Kingdom	37	1074.78
13	Damorin	United Kingdom	22	1044.17
14	ap08960695	Thailand	6	927.84
15	hanzes	Thailand	6	932.06
16	random Sample	United Kingdom	4	876.43
17	tuckyken	Thailand	0	925.25
18	NUDTQi	China	N/A	N/A

Figura 6.1: Resultado final da competição 2-P GVG-AI 2017.

dos jogos selecionados. Também pode-se observar que a queda de desempenho com a utilização das duas técnicas em conjunto foi menos acentuado do que para os experimentos em *single player*. É possível que jogos com características que trazem essa queda no desempenho não estejam inclusos nesta lista de jogos.

Dentre os jogos apresentados, destacam-se o 2 (com um aumeto de 17%), o 8 (com um aumento de 14%) e o 10 (com um aumento de 27%). É possível que o Game 10, o jogo com o maior diferença de desempenho entre os dois controladores, possua características semelhantes ao *Seaquest*, apresentado no conjunto de jogos de jogador único, em que ambas as modificações (menos simulações redundantes e tomada de ações decisivas) podem trazer algum tipo de vantagem para o controlador, ou ao jogo de dois jogadores *Tron*, em que uma das estratégias consegue evitar diversas de derrota no jogo.

É importante lembrar que, diferente dos jogos de jogador único, esta porcentagem de vitórias é em relação à múltiplas execuções de jogos dois a dois em um sistema todos contra todos. Isso significa que pode-se fazer ainda menos suposições sobre a efetividade das técnicas para os jogos propostos, já que existe a possibilidade de não existir outro controlador na competição que implemente técnicas eficientes contra o controlador proposto

(para jogos competitivos) ou técnicas complementares (para jogos colaborativos). Não obstante, apresentar estes resultados torna-se interessante como forma de comparação entre os métodos presentes na literatura.

7 Conclusão e Trabalhos Futuros

Este trabalho apresentou duas novas estratégias para mitigar alguns dos problemas identificados no algoritmo de busca em árvores Monte-Carlo. Os métodos foram implementados e avaliados em jogos tanto de um jogador quanto de dois jogadores. Além disso, os métodos propostos foram utilizados em um controlador participante na competição 2-P GVG-AI.

A primeira modificação propõe um algoritmo para identificação de ações redundantes e uma estratégia para reduzir a ocorrência das mesmas. Através dos experimentos, foi possível observar que a ocorrência dessas ações caíram significativamente, seguidas de um aumento no número de vitórias em alguns jogos onde a movimentação do avatar é um fator importante.

A segunda modificação propôs um método simples porém eficaz de identificar e evitar uma derrota para alguns casos em que o MCTS *vanilla* não é capaz de lidar, e aumentou o número de vitórias para alguns dos jogos onde situações de perigo (como oponentes, projéteis, etc) se aproximam do avatar vindo de múltiplas direções.

O resultado dos experimentos mostraram que os métodos propostos foram capazes de melhorar o desempenho, em relação ao MCTS *vanilla*, para a maioria dos jogos testados. Estes resultados são reforçados tanto pelos experimentos realizados neste trabalho quanto pelos resultados da competição participada, onde o controlador com as modificações apresentadas obteve o 2º lugar no ranking final. Mais importante, foi possível verificar que estes métodos não produzem uma grande sobrecarga na execução do algoritmo, não impactando negativamente o resultado para jogos em que as estratégias não sejam relevantes.

Os resultados sugerem que uma análise mais profunda do impacto do método de identificação e desencorajamento de ações redundantes para diferentes jogos, além do estudo de diferentes parâmetros na penalização de tais ações pode aprimorar ainda mais o desempenho do método. Uma proposta de trabalho futuro é a utilização desta estratégia de identificação de ações redundantes em outras técnicas recorrentes em ambiente GVGP, que poderiam se beneficiar dessa redução de ações redundantes.

Bibliografia

Bansal, B. **Symbolic Logic and Logic Processing**. Laxmi Publications, 2012.

Bontrager, P.; Khalifa, A.; Mendes, A. ; Togelius, J. **Matching games and algorithms for general video game playing**. In: Twelfth Artificial Intelligence and Interactive Digital Entertainment Conference, 2016.

Bouzy, B. Associating domain-dependent knowledge and monte carlo approaches within a go program. **Information Sciences**, v.175, n.4, p. 247–257, 2005.

Browne, C. B.; Powley, E.; Whitehouse, D.; Lucas, S. M.; Cowling, P. I.; Rohlfshagen, P.; Tavener, S.; Perez, D.; Samothrakis, S. ; Colton, S. 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.

Campbell, M.; Hoane, A. J. ; Hsu, F.-h. Deep blue. **Artificial intelligence**, v.134, n.1, p. 57–83, 2002.

Chaslot, G.; Saito, J.-T.; Bouzy, B.; Uiterwijk, J. ; Van Den Herik, H. J. **Monte-carlo strategies for computer go**. In: Proceedings of the 18th BeNeLux Conference on Artificial Intelligence, Namur, Belgium, p. 83–91, 2006.

Chaslot, G.; Bakkes, S.; Szita, I. ; Spronck, P. **Monte-carlo tree search: A new framework for game ai.** In: AIIDE, 2008.

Chaslot, G. Monte-carlo tree search. **Maastricht: Universiteit Maastricht**, 2010.

Champandard, A. J. Monte-carlo tree search in total war: Rome ii's campaign ai. **AIGameDev. com: http://aigamedev. com/open/coverage/mcts-rome-ii**, 2014.

Clune, J. **Heuristic evaluation functions for general game playing**. In: AAAI, volume 7, p. 1134–1139, 2007.

Coulom, R. **Efficient selectivity and backup operators in monte-carlo tree search**. In: International Conference on Computers and Games, p. 72–83. Springer, 2006.

Ebner, M.; Levine, J.; Lucas, S. M.; Schaul, T.; Thompson, T. ; Togelius, J. Towards a video game description language. **Dagstuhl Follow-Ups**, v.6, 2013.

Finnsson, H.; Björnsson, Y. **Simulation-based approach to general game playing.** In: AAAI, volume 8, p. 259–264, 2008.

Frydenberg, F.; Andersen, K. R.; Risi, S. ; Togelius, J. **Investigating mcts modifications in general video game playing**. In: 2015 IEEE Conference on Computational Intelligence and Games (CIG), p. 107–113. IEEE, 2015.

Gaina, R. D.; Pérez-Liébana, D. ; Lucas, S. M. **General video game for 2 players: framework and competition**. In: Computer Science and Electronic Engineering (CEEC), 2016 8th, p. 186–191. IEEE, 2016.

Gaina, R. D.; Liu, J.; Lucas, S. M. ; Pérez-Liébana, D. **Analysis of vanilla rolling horizon evolution parameters in general video game playing**. In: European Conference on the Applications of Evolutionary Computation, p. 418–434. Springer, 2017.

Genesereth, M.; Love, N. ; Pell, B. General game playing: Overview of the aaai competition. **AI magazine**, v.26, n.2, p. 62, 2005.

Genesereth, M.; Björnsson, Y. The international general game playing competition. **AI Magazine**, v.34, n.2, p. 107, 2013.

Kocsis, L.; Szepesvári, C. **Bandit based monte-carlo planning**. In: European conference on machine learning, p. 282–293. Springer, 2006.

Kuhlmann, G.; Stone, P. **Automatic heuristic construction in a complete general game player**. In: AAAI, volume 6, p. 1457–1462, 2006.

Levine, J.; Congdon, C. B.; Ebner, M.; Kendall, G.; Lucas, S. M.; Miikkulainen, R.; Schaul, T. ; Thompson, T. General video game playing. **Dagstuhl Follow-Ups**, v.6, 2013.

Linden, R. **Algoritmos genéticos (2a ediçao)**. Brasport, 2008.

Love, N.; Hinrichs, T.; Haley, D.; Schkufza, E. ; Genesereth, M. **General game playing: Game description language specification**, 2008.

Moriarty, D. E.; Schultz, A. C. ; Grefenstette, J. J. Evolutionary algorithms for reinforcement learning. **J. Artif. Intell. Res.(JAIR)**, v.11, p. 241–276, 1999.

Munos, S. G. W.; Teytaud, O. Modification of uct with patterns in monte-carlo go. **Technical Report RR-6062**, v.32, p. 30–56, 2006.

Nelson, M. J. **Investigating vanilla mcts scaling on the gvg-ai game corpus**. In: Computational Intelligence and Games (CIG), 2016 IEEE Conference on, p. 1–7. IEEE, 2016.

Perez, D.; Samothrakis, S. ; Lucas, S. **Knowledge-based fast evolutionary mcts for general video game playing**. In: 2014 IEEE Conference on Computational Intelligence and Games, p. 1–8. IEEE, 2014.

Perez, D.; Samothrakis, S.; Togelius, J.; Schaul, T.; Lucas, S.; Couëtoux, A.; Lee, J.; Lim, C.-U. ; Thompson, T. The 2014 general video game playing competition. **IEEE Transactions on Computational Intelligence and AI in Games**, 2015A.

Perez Liebana, D.; Dieskau, J.; Hunermund, M.; Mostaghim, S. ; Lucas, S. **Open loop search for general video game playing**. In: Proceedings of the 2015 Annual Conference on Genetic and Evolutionary Computation, p. 337–344. ACM, 2015B.

Perez-Liebana, D.; Samothrakis, S.; Togelius, J.; Lucas, S. M. ; Schaul, T. **General video game ai: Competition, challenges and opportunities**. In: Thirtieth AAAI Conference on Artificial Intelligence, 2016.

Pitrat, J. **Realization of a general game-playing program.** In: IFIP congress (2), p. 1570–1574, 1968.

Ross, B. **General Video Game Playing with Goal Orientation**. 2014. Tese de Doutorado - Master's thesis, University of Strathclyde.

Russell, S. J.; Norvig, P.; Canny, J. F.; Malik, J. M. ; Edwards, D. D. **Artificial intelligence: a modern approach**, volume 2. Prentice hall Upper Saddle River, 2003, 4–5p.

Russell, S. J.; Norvig, P.; Canny, J. F.; Malik, J. M. ; Edwards, D. D. **Artificial intelligence: a modern approach**, volume 2. Prentice hall Upper Saddle River, 2003, 122–123p.

Samuel, A. L. Some studies in machine learning using the game of checkers. **IBM Journal of research and development**, v.3, n.3, p. 210–229, 1959.

Samothrakis, S.; Perez-Liebana, D.; Lucas, S. M. ; Fasli, M. **Neuroevolution for general video game playing**. In: 2015 IEEE Conference on Computational Intelligence and Games (CIG), p. 200–207. IEEE, 2015.

Schiffel, S.; Thielscher, M. **Fluxplayer: A successful general game player**. In: AAAI, volume 7, p. 1191–1196, 2007.

Schiffel, S.; Thielscher, M. **Automated theorem proving for general game playing.** In: IJCAI, p. 911–916. Citeseer, 2009.

Schaul, T. **A video game description language for model-based or interactive learning**. In: Computational Intelligence in Games (CIG), 2013 IEEE Conference on, p. 1–8. IEEE, 2013.

Silver, D.; Huang, A.; Maddison, C. J.; Guez, A.; Sifre, L.; Van Den Driessche, G.; Schrittwieser, J.; Antonoglou, I.; Panneershelvam, V.; Lanctot, M. ; others. Mastering the game of go with deep neural networks and tree search. **Nature**, v.529, n.7587, p. 484–489, 2016.

Soemers, D. J.; Sironi, C. F.; Schuster, T. ; Winands, M. H. **Enhancements for real-time monte-carlo tree search in general video game playing**. In: Computational Intelligence and Games (CIG), 2016 IEEE Conference on, p. 1–8. IEEE, 2016.