RECONHECIMENTO ESPACIAL POR ANÁLISES LÉXICA E SINTÁTICA DE ESTRUTURAS DE VOXELS PARA BOTS DE MINECRAFT

Leonardo Oliveira Santos

Dissertação de Mestrado apresentada ao Programa de Pós-graduação em Engenharia de Sistemas e Computação, COPPE, da Universidade Federal do Rio de Janeiro, como parte dos requisitos necessários à obtenção do título de Mestre em Engenharia de Sistemas e Computação.

Orientador: Geraldo Bonorino Xexéo

Rio de Janeiro

Setembro de 2015

RECONHECIMENTO ESPACIAL POR ANÁLISES LÉXICA E SINTÁTICA DE ESTRUTURAS DE VOXELS PARA BOTS DE MINECRAFT

Leonardo Oliveira Santos

DISSERTAÇÃO SUBMETIDA AO CORPO DOCENTE DO INSTITUTO ALBERTO LUIZ COIMBRA DE PÓS-GRADUAÇÃO E PESQUISA DE ENGENHARIA (COPPE) DA UNIVERSIDADE FEDERAL DO RIO DE JANEIRO COMO PARTE DOS REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE MESTRE EM CIÊNCIAS EM ENGENHARIA DE SISTEMAS E COMPUTAÇÃO.

Examinada por:

_____

Prof. Geraldo Bonorino Xexéo, D.Sc.

_____

Prof. Ricardo Guerra Marroquim, D.Sc.

_____

Prof. Esteban Walter Gonzalez Clua, D.Sc.

RIO DE JANEIRO, RJ - BRASIL
SETEMBRO DE 2015

Santos, Leonardo Oliveira

Reconhecimento Espacial por Análises Léxica e Sintática de Estruturas de Voxels para Bots de Minecraft/Leonardo Oliveira Santos. – Rio de Janeiro: UFRJ/COPPE, 2015.

XVI, 90 p.: il.; 29,7 cm.

Orientador: Geraldo Bonorino Xexéo

Dissertação (mestrado) – UFRJ/ COPPE/ Programa de Engenharia de Sistemas e Computação, 2015.

Referências Bibliográficas: p. 80-90.

1.Gramaticas posicionais livres de contexto. 2.Estruturas de Voxels. 3.Bots de Minecraft. I. Xexéo, Geraldo Bonorino. II. Universidade Federal do Rio de Janeiro, COPPE, Programa de Engenharia de Sistemas e Computação. III. Título.

Aos bots,

que agora poderão compreender melhor onde se encontram.

AGRADECIMENTOS

Primeiramente agradeço a meus pais, pelo amor e estrutura que me permitiram chegar aqui.

À minha amada companheira Jéssica Arruda, pelo apoio, incentivo e cobrança. Minha dedicação a este trabalho muito se deve a ela.

Ao professor Geraldo Xexéo, por seus ensinamentos, inspirações, paciência e confiança. Sua gentil orientação me conduziu à conclusão deste trabalho.

Aos amigos Felipe Horta, Talita Lopes e Verônica Taquette, por compartilharem comigo suas experiências no mestrado. Conhecer seus caminhos me deixou mais confiante em trilhar o meu.

Aos amigos da Inoa Sistemas, por sua compreensão na minha ausência e interesse nos projetos viabilizados por esta pesquisa.

Aos demais amigos que se interessaram em entender a pesquisa, apesar da dificuldade, e me incentivaram.

À equipe da secretaria do PESC, por toda a atenção dispensada e dedicação envolvida em manter minha matrícula ativa.

Por fim, agradeço a todos que amam ou ao longo da história amaram o conhecimento e contribuíram para a existência e aperfeiçoamento da pesquisa e da ciência.

Resumo da Dissertação apresentada à COPPE/UFRJ como parte dos requisitos necessários para a obtenção do grau de Mestre em Ciências (M.Sc.)

RECONHECIMENTO ESPACIAL POR ANÁLISES LÉXICA E SINTÁTICA DE ESTRUTURAS DE VOXELS PARA BOTS DE MINECRAFT

Leonardo Oliveira Santos

Setembro/2015

Orientador: Geraldo Bonorino Xexéo

Programa: Engenharia de Sistemas e Computação

Jogos eletrônicos podem representar um ambiente rico para teste e aprimoramento de inteligências artificiais (IAs). De fato, muitos jogos dispõem de IAs para seu funcionamento, e, em alguns casos, personagens na condição de jogadores são controlados por agentes artificiais inteligentes, chamados *bots*. O Minecraft apresenta-se como um dos exemplos promissores até o momento para esse fim. Em seu espaço de *voxels*, cubos unitários dispostos numa grade discreta e regular, forma-se o terreno e algumas estruturas complexas com as quais os *bots* devem lidar para obter progresso no jogo. Um dos aspectos importantes para as operações de um *bot* é o tratamento dos dados presentes na entrada. Nesse contexto, isso significa identificar formações compostas de *voxels* para embasar suas decisões relativas a posicionamento e locomoção. Esta dissertação investiga a viabilidade de tratar *voxels* como símbolos de uma 'linguagem tridimensional' para que estes sejam processados por analisadores léxicos e sintáticos de gramáticas tridimensionais. Conclui-se que gramáticas posicionais livres de contexto são capazes de realizar tal tarefa em conjunto com o proposto um analisador léxico espacial; são ainda apresentadas gramáticas e relações posicionais para identificação de objetos como árvores, casas, poços d'água e plantações no jogo.

Abstract of Dissertation presented to COPPE/UFRJ as a partial fulfilment of the requirements for the degree of Master of Science (M.Sc.)

SPACE RECOGNITION BY LEXICAL AND SYNTACTIC ANALYSIS OF VOXELS STRUCTRES FOR MINECRAFT BOTS

Leonardo Oliveira Santos

September/2015

Advisor: Geraldo Bonorino Xexéo

Department: Computer Science and System Engineering

Video games can be a rich environment for testing and improvement of artificial intelligence (AI). In fact, many games have AIs for its operation and in some cases, players in the condition of game characters are controlled by intelligent artificial agents, called bots. Minecraft is presented as one of the promising examples to this date for that purpose. In its voxel space, unit cubes arranged in a discreet and regular grid, form the terrain and some complex structures with which the bots must deal to make progress in the game. One of the important aspects for a bot operations is the processing of input data. And in that context, this means identifying formations composed of voxels to base its decisions on positioning and mobility. This dissertation investigates the feasibility of treating voxels as symbols of a three-dimensional language for which they are processed by lexical and syntactic analyzers three-dimensional grammars. It follows that context-free grammars positional are able to accomplish such a task in conjunction with the proposed lexical analyzer space. Also presents grammars and positional relations to identify objects such as trees, houses, water wells and plantations in the game.

Índice

Índice de Figuras

Índice de Tabelas

Índice de Definições

Índice de Algoritmos

Capítulo 1 - Introdução

Na pesquisa e desenvolvimento na área de inteligência artificial (IA), um aspecto particularmente importante é a análise e tratamento do contexto (RICH, 1983). Antes que uma decisão seja tomada e uma ação seja executada, um agente inteligente deve identificar o cenário e a situação em que se encontra. Se o cenário é formado por um objeto, analisar a entrada inclui reconhecer os objetos presentes. Em geral, o termo ‘reconhecimento de objetos’ é associado à análise de imagens bidimensionais e detecção de corpos físicos. As técnicas usadas incluem a análise de formas, movimento, vetores e até técnicas baseadas no córtex visual humano (BELONGIE; MALIK; PUZICHA, 2002; LOWE, 1987; RIESENHUBER; POGGIO, 1999). Um conjunto de ferramentas com propósito semelhante a esse é o conjunto de famílias de analisadores gramaticais, usados no reconhecimento de sequências de símbolos e classificação desta como pertencente ou não da linguagem definida por uma dada gramática. Desde seu surgimento, analisadores de gramáticas têm tido sua capacidade de análise estendida em vários níveis de abstração, permitindo-os trabalhar com formatos de entrada não lineares, como figuras, formas e espaços. Assim, seu uso aplicado pela IA no reconhecimento de objetos é um caminho a ser investigado. A percepção desta existência desse caminho inexplorado inspirou a realização deste trabalho.

1.1 Motivação

Um notável exemplo da aplicação de inteligências artificiais pode ser encontrado no caso dos jogos eletrônicos. Frequentemente a existência de um objeto com IA interagindo com os jogadores humanos torna a experiência do jogo mais interessante e aumenta seu potencial de entretenimento. Quando uma IA ocupa a posição de um jogador do jogo, a esta é dado o nome *bot* (abreviação de *robot*, robô em inglês) (KITANO et al., 1995, 1998). A pesquisa em inteligências artificiais muito contribui (ao mesmo tempo se beneficia) com o desenvolvimento de *bots* para jogos. Quanto mais complexo o universo de um jogo, maior é capacidade exigida de seus jogadores. Isso se aplica tanto a jogadores humanos quanto a *bots*. Por outro lado, essa maior complexidade permite que as IAs dos *bots* se desenvolvam com maior liberdade e tenham, assim, mais espaço pra se

desenvolverem. Afinal, um *bot* de jogo-da-velha, por exemplo, tem menos recursos e menos espaço para expressar complexidade do que o de um jogo mais complexo, como um de xadrez.

Um recente exemplo de ambiente rico em possibilidades é o jogo eletrônico Minecraft (DUNCAN, 2011; GOLDBERG; LARSSON, 2013), ilustrado na Figura 1-1. O jogo tem como base um pequeno conjunto de regras simples que regem o funcionamento do espaço, seus objetos e jogadores. Seu objetivo é simular a realidade natural de uma forma reduzida e simbólica. O principal modo de jogo é o modo 'sobrevivência', em que o jogador deve vencer as adversidades naturais e permanecer jogando pelo maior tempo possível. Os cenários são formados por blocos cúbicos de mesmo tamanho organizados numa grade uniforme. Blocos de determinados tipos posicionados corretamente formam estruturas como árvores ou casas. Para conseguir sobreviver, os jogadores devem ser capazes de se orientar e locomover no espaço em busca de recursos ou fugindo de inimigos.

Figura 1-1 Um cenário no Minecraft

O Minecraft oferece grande potencial para abrigar o desenvolvimento de *bots* inteligentes o bastante para sobreviverem por conta própria e usufruírem as possibilidades que o jogo oferece. Apesar disso, pouco foi desenvolvido até o momento nesse sentido. Em geral os *bots* de Minecraft atuam como assessores sem inteligência de jogadores humanos. Para que existam *bots* capazes de sobreviver autonomamente no jogo, é necessário que estes compreendam seus arredores no tocante ao terreno e as estruturas de blocos que o ocupam. Sem essa capacidade, eles acabarão por morrer sob o ataque de um

inimigo ou de fome. Entretanto, dispondo dessa capacidade, o desenvolvimento e aperfeiçoamento de IAs que sobrevivam no jogo é amplamente facilitado, pois, além dos dados individuais dos blocos próximos, a IA de um *bot* terá à sua disposição também as informações de quais estruturas são formadas por estes blocos. Motiva este trabalho contribuir para o desenvolvimento de IAs em *bots* em jogos, em especial no Minecraft, como forma de fomentar a pesquisa na área e aumentar o entendimento humano sobre as formas de inteligência possíveis. E tem-se como meio escolhido o processamento e análise das informações dos blocos a fim de identificar conjuntos de blocos como exemplares de tipos de estruturas possíveis no cenário.

1.2 Contextualização

O Minecraft foi publicado no final de 2011 por seu criador, Markus Persson (NOTCH, 2015), e posteriormente desenvolvido pela empresa sueca (MOJANG, 2015). O jogo teve grande sucesso com cerca de 20 (vinte) milhões de cópias vendidas pelo mundo até o momento[1]. Em 2014, a Mojang foi adquirida pela empresa estadunidense Microsoft (MICROSOFT, 2014; OWEN, 2014) por cerca de US$ 2,5 bilhões (dois bilhões e meio de dólares americanos). Após a aquisição, Markus Persson e os demais fundadores deixaram a empresa.

O Minecraft é um jogo multiplataforma e pode ser jogado em todos os sistemas operacionais de amplo uso, incluindo computadores pessoais, celulares e consoles de videogame. Ele possui uma arquitetura cliente-servidor (BERSON; OTHERS, 1992) sobre a rede, comportando um volume grandioso de jogadores simultâneos em cada instância de servidor. Segundo (MCSTATS.ORG, 2015), existem até o momento cerca de 130 mil servidores ativos e cerca de 520 mil jogadores simultâneos em horários de pico.

A ambientação do jogo se dá em um espaço tridimensional preenchido por blocos cúbicos organizados numa grade discreta, regular e unitária. Formalmente, eles são uma representação do conceito de *voxels*. Em (CRUMLEY; MARAIAS; GAIN, 2012), um *voxel* é definido como o análogo tridimensional de um *pixel*. O termo vem da contração das palavras ‘*volumetric picture element’* (inglês para ‘elemento volumétrico de figura’, em tradução livre). Assim como *pixels*, os *voxels* são dispostos regularmente no espaço e

[1] Fonte: https://minecraft.net/stats

carregam informações associadas a eles. A *pixels* são atribuídas, em geral, informações de cor e luminosidade. A *voxels* podem ser também atribuídas informações como natureza, rigidez, fluidez, etc. Entretanto, é importante ressaltar que *pixels* e *voxels* são apenas abstrações que sintetizam e discretizam uma data região (SMITH, 1995). Interessam no escopo deste trabalho suas representações como elementos discretos, a fim de que estes sejam tratados como símbolos lidos pelos analisadores gramaticais.

No Minecraft, os blocos são implementações do conceito de *voxel*. Eles formam objetos do cenário e paisagens, que simulam biomas naturais, como florestas, desertos, montanhas nevadas, arquipélagos, planícies, praias e muitos outros. Entre as formações de cenário mais comuns encontram-se diferentes tipos de árvores, cachoeiras, cavernas e vilarejos, formados por estruturas como casas, torres, plantações, postes, ruas e praças, como ilustrado nas figuras 1-2 e 1-3 abaixo.

Figura 1-2 Caverna no Minecraft

Figura 1-3 Vilarejo no Minecraft

Parte do sucesso de público do jogo advém da possibilidade de os jogadores construírem estruturas livremente, segundo sua vontade e a partir dos recursos à sua disposição. Isso possibilita a existência de uma variedade incontável de estruturas espalhadas pelos servidores do jogo.

1.3 Problema

As estruturas sobre o terreno do Minecraft são compostas puramente pela disposição adequada dos blocos que as compõem. Não há nenhuma representação formal de estrutura como uma entidade do jogo. Jogadores humanos são capazes de reconhecer e classificar uma estrutura por conta de sua experiência com o mundo real. É provável que o reconhecimento envolva percepção de formas, texturas em cada parte e as relações posicionais entre as partes de cada estrutura. Para que um *bot* também possa reconhecer estruturas, é preciso processar e analisar os blocos dispostos nas redondezas. Independentemente da estratégia utilizada, é sabido que a informação disponível para entrada está disposta na forma de blocos individuais, e a informação desejada como saída é a localização de estruturas reconhecidas e classificadas como exemplares de uma dada classe. Além disso, ainda é preciso adotar uma forma de varredura eficiente do espaço tridimensional de entrada.

Numa analogia entre o problema abordado neste trabalho e o problema resolvido pelos compiladores e analisadores gramaticais, um *voxel* está para uma estrutura de *voxels* assim como uma letra está para um texto. A diferença é a dimensionalidade do problema.

Um corpo de texto ocupa um espaço unidimensional discreto. Cada caractere ocupa uma posição numa linha de posições regulares. Mesmo uma quebra-de-linha é representada simplesmente como um caractere ou dois, dependendo da convenção adotada. A forma como esses caracteres se dispõem ao longo das linhas é descrita pela gramática da linguagem utilizada. De forma análoga, porém, em três dimensões, este trabalho entende que a disposição de *voxels* numa estrutura no espaço pode ser descrita por uma gramática e pertencer, assim, a uma linguagem de estruturas de *voxels*. Uma diferença entre esses dois casos, além da dimensionalidade, é que os textos sobre as linhas ocupam um espaço que tem início definido e podem ser indefinidamente longos; enquanto o espaço tridimensional das estruturas de *voxels* não possui necessariamente início e fim definidos.

1.4 Objetivo

De forma geral, o objetivo deste trabalho é encontrar um método para o reconhecimento e classificação dos objetos formados por *voxels*. Considerada a implementação de *voxel* do Minecraft, esse reconhecimento refere-se um conjunto de blocos como pertencente ou não a uma dada classe de estruturas do jogo.

Pela natureza do problema, e sobretudo sua semelhança com o problema da interpretação e compilação de linguagens regulares, faz-se relevante investigar a possibilidade de analisadores léxicos e sintáticos operarem sobre espaços de dimensões maiores que um, neste caso, tridimensionais. Especificamente, portanto, este trabalho objetiva avaliar a viabilidade do uso de analisadores LR (AHO; AHO, 2007) para o reconhecimento e classificação de estruturas sobre um espaço tridimensional.

1.5 Metodologia

A primeira etapa desta pesquisa foi conduzir um trabalho investigativo por meio de uma revisão bibliográfica buscando caracterizar o atual estado de pesquisa em tópicos envolvendo gramáticas bidimensionais e tridimensionais, além de *bots* e IAs em jogos e, ainda, *bots* no Minecraft. Vale ainda notar que técnicas de aprendizagem de máquina seriam um caminho viável a ser investigado, entretanto, envolveriam uma etapa prévia de treinamento dos analisadores e obteriam um resultado menos preciso do que as análises gramaticais. Assim sendo, a pesquisa se limitou a gramáticas bidimensionais e tridimensionais. A fim de suportar o processo de análise e síntese do material reunido, foi

construído um mapa conceitual representando os esforços envolvidos em cada área de interesse. Tais mapas foram importantes para tornar mais visual e concisa a compreensão dos resultados. No tocante às gramáticas, a busca era por um modelo que satisfizesse os requisitos do problema, seja por meio de uma gramática tridimensional, seja pela extensão a ser feita sobre uma gramática bidimensional.

Dentre as técnicas disponíveis, as gramáticas posicionais livres de contexto de (COSTAGLIOLA; CHANG, 1999) foram adotadas na solução proposta pois melhor se adequaram ao problema tanto no aspecto teórico quanto no tecnológico. De modo a viabilizar a análise de estruturas tridimensionais, foi desenvolvido um analisador léxico espacial, apresentado nesta dissertação. Adicionalmente, foram concebidas também relações posicionais básicas e gramáticas posicionais livres de contexto capazes de reconhecer certas estruturas comumente encontradas no Minecraft.

Por fim, um *bot* simples capaz de responder a comandos básicos e munido dos analisadores de tais gramáticas foi também desenvolvido. Por meio deste *bot* objetiva-se ilustrar o funcionamento das gramáticas posicionais livres de contexto em conjunto com o analisador léxico espacial, além de medir sua performance.

1.6 Organização

O presente texto estrutura-se em nove capítulos, incluindo este primeiro de introdução, que apresenta a motivação, o contexto em que o trabalho foi realizado, o problema abordado, o objetivo almejado e a metodologia usada na pesquisa.

O capítulo 2 trata dos trabalhos correlatos. São apresentados os demais trabalhos e pesquisas envolvendo gramáticas bidimensionais e tridimensionais, inteligências artificiais e *bots* em jogos e, por fim, *bots* específicos de Minecraft.

No capítulo 3 são reapresentadas as gramáticas posicionais livres de contexto. Ele traz uma reprodução das definições e algoritmos que constituem o conceito. Seu objetivo é viabilizar uma fundamentação autocontida deste trabalho, necessária ao seu entendimento.

O jogo eletrônico Minecraft é melhor detalhado no capítulo 4, no qual se apresenta-se o contexto em que este trabalho se insere. Além disso, são detalhadas também as características do jogo e dos meios para criação de *bots* de Minecraft.

No capítulo 5 é descrita a proposta de solução deste trabalho. Nele estão incluídas as definições de relações posicionais básicas usadas na identificação de algumas

estruturas e, sobretudo, do analisador léxico espacial, que viabiliza a leitura prática do espaço de entrada para realização da análise sintática. Um exemplo detalhado do funcionamento do analisador léxico espacial também está presente.

O capítulo 6 contém uma lista de exemplos de estruturas comumente encontradas no Minecraft e que foram consideradas relevantes para a ilustração do funcionamento da solução proposta. Para cada exemplo são definidas suas respectivas gramáticas posicionais livres de contexto e as relações posicionais necessárias à condução da leitura do espaço tridimensional de entrada.

Detalhes sobre a implementação do analisador léxico espacial estão presentes no capítulo 7. Nele também está incluída a descrição da interface de gerência da leitura da entrada e a de um simples *bot*. Este é capaz de reconhecer o espaço à sua volta por meio dos analisadores das gramáticas apresentadas no capítulo 6.

Para medir a performance, três experimentos foram conduzidos num ambiente de testes. Os resultados são apresentados e discutidos no capítulo 8. O objetivo dos experimentos é verificar a viabilidade de utilização da solução proposta para cenários reais, em que um *bot* analise e reconheça estruturas próximas.

Por fim, o último capítulo conclui este trabalho sumarizando as contribuições apresentadas. São discutidos também os trabalhos futuros que podem ser vislumbrados no momento.

Capítulo 2 - Trabalhos Correlatos

Muito foi desenvolvido sobre gramáticas computacionais desde seu surgimento na década de 1950. Dentre os tipos de gramáticas não-convencionais estão incluídas, por exemplo, as de grafos, as de formas, as de analisadores aumentados e as de generalizados. Em geral, tais trabalhos não viabilizam uma abstração compatível com o espaço de *voxels* que é abordado neste texto. Serão relatados abaixo alguns dos trabalhos relacionados a este no tocante ao uso de gramáticas para a leitura de dados em dimensões superiores a um. Posteriormente apresentam-se os trabalhos relacionados a *bot*s em jogos e então, mais especificamente, no Minecraft.

2.1 Gramáticas Bidimensionais

A pesquisa sobre gramáticas bidimensionais apresenta um possível caminho à solução do problema tratado nesta dissertação caso seja considerada uma extrapolação da técnica bidimensional para suportar três dimensões. Apesar da existência de várias propostas de gramáticas bidimensionais, nenhuma teoria completa foi formada ainda. Para ser considerada gramática bidimensional no escopo deste texto, uma dada gramática deve ser capaz de gerar produtos que obrigatoriamente usem pelo menos dois graus de liberdade para serem descritos. A linha de pesquisa adotada por este trabalho foi introduzida por (CHANG, 1971) sob o nome de gramáticas de processamento de figuras. Elas são uma extensão tabelar das gramáticas de estruturas em frases unidimensionais, e seus trabalhos derivados serão tratados posteriormente neste texto.

Em uma linha independente, um dos trabalhos mais relevantes no assunto é o de (TOMITA, 1991), no qual é definida a análise de linguagens bidimensionais pelo uso de seu algoritmo de análise de gramáticas aumentadas livres de contexto descritas em (TOMITA, 1987). A técnica consiste em substituir a tabela GOTO dos *parsers* LR por duas tabelas GOTO-Right e GOTO-Down para controlar respectivamente os movimentos para a direita e para baixo da posição de leitura. Entretanto, essa abordagem limitada nos sentidos de leitura em cada dimensão requer que uma figura seja apresentada sempre na mesma orientação, não sendo admitidas rotações, por exemplo.

Com uma abordagem diferente, (WITTENBURG; WEITZMAN, 1998) descrevem o uso de gramáticas relacionais com atributos utilizadas para a interpretação de modelagem de processos de negócios. Essas gramáticas operam construindo um grafo ao longo de suas derivações. A cada aplicação de uma regra de produção, as ditas 'relações associadas' a uma regra são aplicadas sobre os elementos do lado direito da regra e descrevem arestas que devem ser criadas entre os nós representativos de elementos. A análise é feita pela agregação de subgrafos até que se chegue ao símbolo inicial, raiz da árvore de derivação. Essa técnica é muito semelhante à que foi adotada neste trabalho. Contudo, ela requer que as relações entre os elementos estejam previamente estabelecidas. Não é possível aplicá-la à leitura de blocos ocupando o espaço sem relações explicitas entre eles.

Em outra linha de pesquisa, (SUBRAMANIAN et al., 2008, 2009) descreve modelos de linguagens e gramáticas puras para figuras bidimensionais (P2DCFG). Estas são figuras retangulares análogas a matrizes. Isso permite-as serem representadas como concatenação horizontal de vetores coluna de mesmo tamanho ou concatenação vertical de vetores linha de mesmo tamanho. As regras de produção dessas gramáticas são divididas em duas categorias: produções de linhas e produções de colunas. Suas derivações consistem em substituir linhas ou colunas de uma figura através de uma regra de produção da categoria respectiva. O título de "puras" vem do fato de que não existem símbolos não-terminais, ou seja, ambos os lados das regras de produção são compostos por símbolos terminais (MAURER; SALOMAA; WOOD, 1980). Apesar de representar uma família relevante entre as gramáticas bidimensionais, não existe até o momento uma técnica de análise e interpretação de figuras das linguagens descritas por estas gramáticas.

Em (PRUŠA, 2001), é descrita a geração de figuras bidimensionais por meio de uma extensão para duas dimensões das gramáticas livres de contexto. Sua análise é feita através de uma extensão da máquina de Turing chamada 'Forgetting Automata' (em tradução livre do inglês, 'autômato com esquecimento')(PETR JIRICKA, 1999). As definições sobre gramáticas são semelhantes às de (SUBRAMANIAN et al., 2008, 2009), exceto pela presença de símbolo não-terminais, o que não as caracteriza como "puras". Os autômatos com esquecimento são máquinas de Turing que podem sobrescrever uma posição da fita apenas com um símbolo especial (diz-se que esta posição foi esquecida), enquanto a informação é armazenada em blocos para permitir a reconstrução da figura de entrada. Esse autômato foi estendido para operar sobre uma fita bidimensional e usado para reconhecer figuras bidimensionais. Todavia, o uso desses autômatos não se mostra

adequado para interpretação de linguagens bidimensionais livres de contexto, como o próprio (PRUŠA, 2001) indica.

Em (MARTINOVIC; VAN GOOL, 2013a, 2013b) foram usadas fachadas prediais para buscar gramáticas estocásticas capazes de generalizar as formações urbanísticas de um dado conjunto de entrada. Eles apresentam as gramáticas bidimensionais estocásticas livres de contexto e com atributos (2D-ASCFG), que são tuplas formadas pelos tradicionais conjuntos de elementos terminais e não-terminais do símbolo inicial e do conjunto das regras de produção, além dos propostos conjuntos de probabilidades e de atributos sobre as regras de produção. Um modelo bayesiano foi utilizado para efetuar a busca pelas gramáticas. A análise é feita por meio de uma extensão bidimensional (STOLCKE, 1994) do algoritmo de (EARLEY, 1970). Por utilizar técnicas estocásticas, essa abordagem requer um volume de estruturas de exemplo, o que não está disponível no caso tratado por esta dissertação.

Uma abordagem para o reconhecimento de expressões matemáticas manuscritas foi definida em (MACLEAN; LABAHN, 2013). O reconhecimento dos símbolos é feito por *fuzzy sets* (ZADEH, 1965), nos quais os símbolos são posteriormente tratados pelas gramáticas posicionais de (COSTAGLIOLA et al., 1998), que serão melhor apresentadas no próximo capítulo.

2.2 Gramáticas Tridimensionais

Assim como no caso das gramáticas bidimensionais, os diversos tipos de gramáticas tridimensionais ainda não formam uma teoria completa. Muitas das gramáticas tridimensionais são fruto de extrapolação de dimensionalidade das bidimensionais. Em alguns casos, isso consequentemente leva também a gramáticas de dimensão superiores a três.

Um dos tipos de gramática bidimensional expandida para três dimensões são as gramáticas de formas. Seu uso foi introduzido por (STINY; GIPS, 1971) na geração de esculturas como extensão das gramáticas de formas para pinturas. Elas foram usadas para comparar estruturas de blocos cúbicos uniformes e adjacentes visualizadas por ângulos diferentes (GIPS, 1974). As imagens das estruturas são pré-processadas, e os vértices visíveis dos blocos são categorizados. Posteriormente, é aplicada a análise sintática da estrutura de vértices para produzir um modelo da estrutura. Os modelos são, então, comparados para determinar se as estruturas de entrada têm a mesma forma e estão sendo

vistas por perspectivas diferentes. Apesar de essa abordagem fazer uso de analisadores gramaticais, ela requer uma etapa de visualização de imagens, o que, no caso tratado nesta dissertação envolveria uma etapa de '*renderização'*, considerada desnecessária. Entre os demais trabalhos de gramáticas de formas, (CRUMLEY; MARAIAS; GAIN, 2012) usa-as para geração de objetos que são posteriormente *'voxelizados'*. A análise não é abordada. (PIAZZALUNGA; FITZHORN, 1998) discutem e exemplificam uma implementação de gramáticas de formas. Em (TEBOUL, 2011), gramáticas de formas são usadas para a análise e interpretação de imagens de prédios. (KRISHNAMURTI, 2013) faz uma revisão de gramáticas de formas e demonstrada sua aplicação em arquitetura. (LIN; FU, 1984) introduz gramáticas de superfícies, nas quais as regras de produção funcionam como relacionamento estrutural entre as superfícies. Em todos esses casos, as gramáticas referem-se a polígonos posicionados em espaços contínuos, o que não é compatível com as características do problema tratado nesta dissertação, ou seja, *voxels* como elementos discretos individuais.

Gramáticas de vetores tridimensionais são usadas por (SIROMONEY; KRITHIVASAN; SIROMONEY, 1973) para geração de cristais simétricos. Em (WANG, 1991), é apresentada uma forma de representação de objetos tridimensionais por gramáticas de vetores tridimensionais, e a geração dos padrões é feita de forma paralelizada. A análise desses padrões, porém, não é abordada.

As gramáticas 'Plex' são apresentadas por (PENG; YAMAMOTO; AOKI, 1990) como conjuntos de símbolos interconectados em múltiplas direções. Sua análise é feita com uma extensão do algoritmo de (EARLEY, 1970). As gramáticas de grafos (ROZENBERG, 1997) são outro caso de extrapolação de duas para três dimensões. Estas foram usadas em (PICKEM, 2011) na reconfiguração tridimensional de robôs modulares. Assim como no caso das gramáticas relacionais de (WITTENBURG; WEITZMAN, 1998), são necessárias relações previamente estabelecidas entre os nós, o que não ocorre no problema abordado nesta dissertação.

Gramáticas de relação são apresentadas por (CRIMI et al., 1990) como extensão das gramáticas de processamento de figuras de (CHANG, 1971) e usadas na descrição da sintaxe de linhas horizontais e gráficos de diagramas de estado. Esse linha de pesquisa, por focar na leitura de grafos, em que há relações entre os nós, assemelha-se com a adotada em (WITTENBURG; WEITZMAN, 1998) e, portanto, não se adequa a espaços em que essas relações não são explícitas.

As gramáticas de (LINDENMAYER, 1971) descrevem estruturas autossemelhantes, como fractais. Nelas as regras de produção são aplicadas ao axioma, e posteriormente a sequência formada é transformada numa estrutura geométrica. Essas gramáticas foram usadas por (MUHAR, 2001) para geração de vegetação em terrenos simulados. Em (JACOB; MAMMEN, 2007), elas são estendidas para gramáticas de exame, em que mais de um agente processa a sequência de símbolos em uma estrutura geométrica.

Entre tópicos relacionados a gramáticas tridimensionais, porém com foco distinto, encontram-se as expressões regulares tridimensionais apresentadas por (WURZER; MARTENS; BÜHLER, 2013) e usadas na busca por geometrias recorrentes em malhas para arquitetura predial. Outras formas de se referir a gramáticas tridimensionais porém com significado diferente do utilizado neste trabalho pode ser encontrado em (DE VRIES, 2003; SOCHER et al., 2013; STEEN, 2008). Estes trabalhos compreendem as dimensões da gramática não de forma espacial, mas metafórica, referindo-se a dimensões de significado dos termos em um sequência linear de símbolos.

2.3 Inteligência Artificial e Bots em Jogos

A pesquisa em inteligência artificial está muitas vezes associada a *bots* de jogos. Estes se beneficiam amplamente dos avanços da pesquisa enquanto oferecem ambiente propício à validação das técnicas em desenvolvimento. A literatura disponível aborda o assunto sob diversas perspectivas, incluindo a detecção e a avaliação de qualidade de *bots*, o aprimoramento de suas funções, além de plataformas de desenvolvimento e execução de testes. Em geral, as implementações de *bots* são específicas para um jogo e pela função desempenhada pelo *bot*. Assim sendo, costumam servir apenas como inspiração para o desenvolvimento de *bots* em outros jogos. Esta seção tem por objetivo exatamente ilustrar as possibilidades de uso de IAs em jogos a fim de balizar um futuro uso da solução proposta neste trabalho num *bot* inteligente de Minecraft.

Tratando de IAs em jogos, porém desassociadas a *bots*, (DOYLE; DEAN, 1996) indicaram os primeiros passos e as direções futuras da pesquisa realizada nessa área. Relatórios e revisões sobre uso de IAs em jogos podem ser encontrados em (ANDERSON, 2003) e (BURO; CHURCHILL, 2012); além disso, eles reforçam a motivação para pesquisa nesta área. Em (CHARLES, 2003) são discutidas as questões e implicações decorrentes da utilização de IAs em jogos. E em (JOHNSON; WILES, 2001)

estuda-se o caso de um jogo de estratégia com IA, o ‘Black and White’, do estúdio Lionhead.

Experimentações e investigações sobre a criação e desenvolvimento de *bots* foram descritos em (LAIRD, 2002; LAIRD; DUCHI, 2000; VAN LENT; LAIRD, 1998). Em (KHOO; ZUBEK, 2002) são descritos dois *bots*, e suas implementações são comparadas e discutidas. Entre os ramos desta área, o do aprimoramento talvez seja o mais pesquisado, tendo em vista o volume de publicações em torno da aplicação de uma nova técnica de IA em *bots* de um determinado jogo. (KHOO et al., 2002) descrevem dois sistemas com base em comportamento para o controle de personagens não jogadores. Já (LE HY et al., 2004) usam programação Bayesiana para reger os comportamentos de seus *bots*. Em (HINDRIKS et al., 2011) é discutido o uso de lógicas BDI (‘Belief, Desire & Intention’, inglês para ‘crença, desejo e intenção’) de alto nível para *bots* em ambientes de tempo real. Modelos baseados no comportamento humano para *bots* são descritos em (SILVERMAN et al., 2006a, 2006b). E a aprendizagem baseada em imitação é descrita em (GORMAN; HUMPHRYS, 2007; THURAU; BAUCKHAGE; SAGERER, 2004). Em (SONI; HINGSTON, 2008; VON AHN; LIU; BLUM, 2006) a avaliação humana é empregada para direcionar a evolução dos bots. Em (SCHAEFFER; BULITKO; BURO, 2008; ZUBEK; KHOO, 2002) são utilizadas diversas técnicas para tornar os bots mais convincentes na percepção de jogadores humanos. A aplicação de campos potenciais multiagente pode ser encontrada em (HAGELBÄCK; JOHANSSON, 2008, 2009) para *bots* de jogos de estratégia em tempo real. Em (BAUMGARTEN; COLTON; MORRIS, 2009; LIM; BAUMGARTEN; COLTON, 2010; PRIESTERJAHN et al., 2006) são experimentadas técnicas variadas, incluindo árvores de decisão, raciocínio baseado em caso, arrefecimento simulado e árvores de comportamento evolutivas. (MCPARTLAND; GALLAGHER, 2008a, 2008b, 2011) utilizam a técnica de reforço para refinar os comportamentos dos *bots*. Em (MUNOZ-AVILA; FISHER, 2004) é proposto o uso de planejamento estratégico executado por um ou mais *bots* numa mesma partida do jogo.

Redes neurais são um tópico recorrente nesse meio, e (MIIKKULAINEN et al., 2006) encorajam seu uso; além disso, apresentam uma revisão geral das técnicas de inteligência computacional utilizadas em jogos. Redes neurais também são aplicadas em *bots* por (BAUCKHAGE; THURAU; SAGERER, 2003; SONI; HINGSTON, 2008; WANG et al., 2009). Algoritmos genéticos ou evolutivos são utilizados em *bots* por (COLE; LOUIS; MILES, 2004; ESPARCIA-ALCÁZAR et al., 2010; FERNANDEZ-ARES et al., 2011; MORA et al., 2010). E a combinação de redes neurais e algoritmos

genéticos, também chamada de neurogenéticos ou neuroevolutivos, é descrita em (DARRYL, 2007; SCHRUM; KARPOV; MIIKKULAINEN, 2011; SPRONCK; SPRINKHUIZEN-KUYPER; POSTMA, 2003). Uma habilidade desejável a *bots* de certos jogos é a capacidade de compreender seus oponentes. Em (SCHADD; BAKKES; SPRONCK, 2007) são utilizados modelos hierarquicamente estruturados para modelar os oponentes dos *bots*. (LAIRD, 2001) usa decomposição dinâmica e hierárquica de tarefas para que os *bots* possam antecipar as ações dos adversários. (HLADKY; BULITKO, 2008) utilizam um modelo de semi-Markov oculto pra prever a posição do oponente e discutem o fato de que geralmente as IAs têm acesso aos dados internos do jogo, o que é geralmente considerado trapaça pelos jogadores humanos.

A pesquisa sobre consciência computacional é aplicada a *bots* em (ARRABALES; LEDEZMA; SANCHIS, 2009) na produção de jogadores mais convincentes. Outro ramo bastante estudado é a identificação ou detecção de *bots* nos jogos (YAMPOLSKIY; GOVINDARAJU, 2008). Nesse sentido, existem trabalhos tratando da identificação a fim de impedir a ação de *bots* onde estes não são desejados. Em contrapartida, também existem os que desejam contribuir para que *bots* passem no teste de Turing, tais como o BotPrize, um prêmio organizado anualmente para incentivar o aprimoramento das inteligências artificiais em *bots*, descrito em (HINGSTON, 2009, 2010). Um dos vencedores pode ser encontrado em (BOYLE, 2012; SCHRUM; KARPOV; MIIKKULAINEN, 2011). Nos casos em que a presença de bots é indesejada num dado ambiente de um jogo, as técnicas utilizadas incluem HOP ('Human Observation Proofs', inglês para 'provas de observação humana') como proposto em (GIANVECCHIO et al., 2009) e modelos de difusão em (WOO; KANG; KIM, 2012). Das plataformas para desenvolvimento e testes de *bots*, tratam (ADOBBATI et al., 2001; AHA; MUÑOZ-AVILA; VAN LENT, 2005; KAMINKA et al., 2002).

2.4 Bots no Minecraft

Dentre os trabalhos sobre IAs em jogos, serão relatados abaixo os envolvidos com *bots* no Minecraft. Em geral estes *bots* são usados por jogadores humanos para a realização de tarefas repetitivas, de pouco valor de entretenimento, porém cujo resultado é de interesse dos jogadores. Assim os *bots* operam recebendo comandos dos jogadores humanos e executando tarefas predefinidas, como varrer o espaço em busca de um determinado material, coletar uma certa quantidade de uma dada substância, executar uma

rotina de escavação, replicar uma estrutura ou construção e até manter vigília de uma região atacando os inimigos que se aproximam.

Em (KELLEY et al., 2015) foi desenvolvida uma biblioteca pra facilitar a criação de *bots* de Minecraft chamada Mineflayer. Ela dispõe de *plug-ins* para contribuir com a navegação dos *bots* (KELLEY, 2013a, 2013b, 2013c), para monitorá-los pela rede (KELLEY, 2013d), e também de uma rotina de busca por blocos (KUBALL, 2013a). Exemplos de *bots* desenvolvidos com base no Mineflayer incluem o ArcherBot (KELLEY, 2013e), rBot (BEAUMONT, 2015) e o HelperBot (KUBALL, 2013b). O primeiro tem por objetivo aceitar e proceder em um duelo de cavalheiros utilizando arco e flechas; os seguintes reúnem diversas funções para assessorar seu jogador humano mestre.

Outra vertente do uso de *bots* em Minecraft é a pesquisa para o aprimoramento de algoritmos de priorização de tarefas para robôs físicos (ABEL et al., 2015; ABEL; TELLEX, 2015; BARTH-MARON et al., 2014). Para solucionar um problema, um robô precisa avaliar as possibilidades e priorizar suas ações. Entretanto, testar e refinar algoritmos de priorização fazendo uso de robôs físico é mais demorado do que simular virtualmente os cenários de teste. Para isso, *bots* de Minecraft estão sendo usados para evoluir esses algoritmos (STACEY, 2015). Além disso, Minecraft também foi usado para ensinar Inteligência Artificial para alunos de graduação (BAYLISS, 2012). Há ainda uma versão modificada (ou *mods*) do Minecraft para favorecer a automação e criação de *bots* em (STORM, 2014).

Capítulo 3 - Gramáticas Posicionais Livres de Contexto

Este capítulo compreende a breve reprodução das definições e algoritmos apresentados em (COSTAGLIOLA; CHANG, 1999) com o intuito de manter autocontido o embasamento técnico deste trabalho.

Gramáticas posicionais livres de contexto são uma extrapolação das gramáticas livres de contexto para dimensões maiores que 1 (um), e sua interpretação é possível estendendo-se o funcionamento dos analisadores gramaticais LR pelo uso de relações posicionais entre seus símbolos para efetuar a leitura de figuras. Um exemplo de figuras bidimensionais seriam as expressões aritméticas. Em (COSTAGLIOLA; CHANG, 1999) é definida a adição de uma nova coluna 'Position' à tabela 'Action-GoTo' como uma extensão dos analisadores sintáticos LR. Assim, é possível guiar a leitura dos símbolos da figura de entrada de acordo com as relações posicionais usadas na derivação. Em (COSTAGLIOLA; CHANG, 1999) também é descrito como utilizar analisadores sintáticos LR já existentes e sem qualquer modificação de seu funcionamento para realizar a tarefa de analisar figuras simbólicas. Seguem abaixo as definições relativas às gramáticas posicionais livres de contexto.

Definição 3-1 Gramáticas Posicionais (COSTAGLIOLA; CHANG, 1999)

Uma gramática posicional livre de contexto pode ser representada por uma 6-tupla (N, T, S, P, POS, PE) onde:

N é um conjunto finito e não vazio de símbolos não terminais,

T é um conjunto finito e não vazio de símbolos terminais,

$N \cap T = \emptyset$,

$S \in N$ é um símbolo inicial,

P é um conjunto finito de produções,

POS é um conjunto finito de identificadores de relações posicionais,

$POS \cap (N \cup T) = \emptyset$,

PE é um Avaliador Posicional

Cada produção em P tem a seguinte forma:

$$A \rightarrow x_1 \, REL_1 \, x_2 \, REL_2 \ldots REL_{m-1} \, x_m \qquad m \geq 1$$

Onde A ∈ N, cada x_i está em N ∪ T e cada REL_i está em POS. ▪

Assim, (COSTAGLIOLA; CHANG, 1999) consideram que cada REL_i dá informação sobre a posição relativa de x_{i+1} a partir de x_i. A partir de agora, o termo "gramática posicional" será usado para descrever gramática posicional livre de contexto, e blocos ou voxels são considerados símbolos terminais. A seguir, as definições de figura simbólica e da função de localização de uma figura:

Definição 3-2 Figura Simbólica (COSTAGLIOLA; CHANG, 1999)
Seja V um conjunto de símbolos, n > 0 um inteiro e P um conjunto de posições no espaço *n*-dimensional. Uma figura simbólica de dimensões *n* em V e P é um mapeamento *p*: P → V. Quando |P| = 1 a figura simbólica se reduz a um símbolo. ▪

Definição 3-3 Localização de uma figura (COSTAGLIOLA; CHANG, 1999)
Seja S um conjunto de todos as figuras simbólicas de dimensão *n* em um vocabulário V e um conjunto de posições P. A localização de uma figura simbólica *p* ∈ S é o mapeamento *location*: S→ P.▪

Tem-se, portanto, que, dada uma figura *f*, a função *location*(*f*) retorna à posição de *f* no espaço. Seguem abaixo alguns exemplos simples de relações posicionais, no espaço bidimensional discreto:

BEFORE = {(f1, f2): f1 e f2 são figuras;
location(f1) = (x, y);
location(f2) = (x', y');
x' = x+1;
y' = y}

UNDER = {(f1, f2): f1 e f2 são figuras;
location(f1) = (x, y);
location(f2) = (x', y');
x' = x;
y' = y+1}

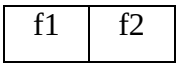

Figura 3-1
BEFORE

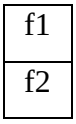

Figura 3-2
UNDER

A convenção utilizada entende a dimensão *x* como horizontal e a *y* como vertical. As figuras 3-1 e 3-2 representam respectivamente as relações posicionais BEFORE e UNDER descritas acima. A transformação de uma sequência de símbolos gerada por uma gramática posicional numa figura é feita por um avaliador posicional, como definido a seguir.

Definição 3-4 Avaliador Posicional (COSTAGLIOLA; CHANG, 1999)
Um avaliador posicional PE (do inglês, Positional Evaluator) é uma função cuja entrada é uma sequência

$$p_1 \, REL_1 \, p_2 \, REL_2 \ldots REL_{m-1} \, p_m \qquad m \geq 1$$

Onde cada p_i é uma figura simbólica e cada REL_i é uma relação posicional; sua saída é uma nova figura constituída por figuras $p_1, p_2, \ldots, p_m$ dispostas no espaço de modo que

$$(p_i , p_{i+1}) \in REL_i \qquad 1 \leq i \leq m\text{-}1.$$

A avaliação de relações posicionais deve ser sequencial da esquerda para a direita. ▪

Dando-se continuidade ao formalismo, seguem abaixo as definições relativas à derivação de figuras por uma gramática posicional.

Definição 3-5 (COSTAGLIOLA; CHANG, 1999)
Escreve-se $\Pi \Rightarrow \Sigma$ se existem Δ, Γ, A, ψ de modo que $\Pi = \Gamma A \Delta$, $A \rightarrow \psi$ é uma produção e

$$\Sigma = \begin{cases} \Gamma\{\psi\}\Delta, & se\ |\psi| > 1 \\ \Gamma\psi\Delta, & se\ |\psi| = 1 \end{cases}$$

Onde '$\{\psi\}$' é considerado como uma sequência literal e não como seu valor; Γ e Δ podem não estar balanceadas, i.e., o número de chaves a esquerda pode não ser igual ao número de chaves a direita. ▪

Isto feito de forma encadeada permite o surgimento de derivações, como é apresentado a seguir:

Definição 3-6 (COSTAGLIOLA; CHANG, 1999)

Escreve-se $\Pi \Rightarrow^* \Sigma$ (Σ é derivado de Π) se existe uma sequência Π_0 , Π_1 ... Π_m ($m \geq 0$) de modo que

$$\Pi = \Pi_0 \Rightarrow \Pi_1 \Rightarrow ... \Rightarrow \Pi_m = \Sigma$$

A sequência $\Pi_{0, ..., } \Pi_m$ é chamada de derivação de Σ a partir de Π. Uma *forma posicional sentencial* é a *string* Π de modo que $S \Rightarrow^* \Pi$. Uma *sentença posicional* é uma forma posicional sentencial que não contenha símbolos não terminais. Uma *forma pictórica* é a avaliação de uma forma posicional sentencial. Uma *figura* é uma forma pictórica com apenas símbolos terminais. A *linguagem pictórica definida por uma gramatica posicional L(PG)* é um conjunto de figuras. ▪

Seguindo a convenção, onde não for definido, usam-se termos em caixa alta para denotar símbolos não-terminais e termos em caixa baixa para símbolos terminais. As relações posicionais serão destacadas em negrito, e os símbolos terminais, em itálico.

Por exemplo, a gramática a seguir gera escadas ascendentes da esquerda para a direita. As relações posicionais UNDER e BEFORE usadas a seguir são as apresentadas acima.

STAIRS → STEP **BEFORE** STAIRS
STAIRS → STEP
STEP → *stringer* **UNDER** *tread*

Uma derivação possível desta gramática seria, por exemplo:

stringer **UNDER** *tread* **BEFORE** *stringer* **UNDER** *tread* **BEFORE** *stringer* **UNDER** *tread*

Para facilitar a visualização, segue abaixo, na Figura 3-3, a disposição dos símbolos após a avaliação posicional:

		tread
	tread	*stringer*
tread	*stringer*	
stringer		

Figura 3-3 Escada

Antes da definição da linearidade, (COSTAGLIOLA; CHANG, 1999) definem as funções abaixo:

Definição 3-7 HEAD e TAIL (COSTAGLIOLA; CHANG, 1999)

Seja p^s a figura resultante da avaliação de uma sentença posicional s, define-se HEAD(p^s) como o símbolo em p^s que inicia s, e TAIL(p^s) como o símbolo em p^s que termina s. ▪

No exemplo anterior, o HEAD da figura é o primeiro bloco *stringer*, e TAIL é o último bloco *tread*.

Definição 3-8 Gramática Posicional Linear (COSTAGLIOLA; CHANG, 1999)

Seja p^s uma figura resultante da avaliação de uma sentença posicional s. Uma gramática posicional LPG (do inglês, Linear Positional Grammar) é linear se, e somente se, a ocorrência de "x REL y" em uma produção da LPG implica que para cada sentença posicional s derivável de x e para cada sentença posicional t derivável de y, com x, y ∈ N ∪ T,

$$(\text{TAIL}(p^s), \text{HEAD}(p^t)) \in \text{REL}$$

Em uma figura obtida de uma execução de $\{p^s \text{ REL } p^t\}$. ▪

Para controlar a finalização da leitura da entrada, é definido o operador ANY abaixo. Sua implementação pode ser feita permitindo que o analisador marque os símbolos visitados e, quando acionado, o ANY apenas verifica a existência de blocos não visitados na entrada.

Definição 3-9 Operador ANY (COSTAGLIOLA; CHANG, 1999)

O operador ANY de fim-da-entrada é uma função cujo valor de retorno será 0(zero) se todos os símbolos da entrada tiverem sido lidos e 'erro' caso contrário. ▪

Um ponto muito importante na construção dos analisadores de LPGs é a implementação de operador posicional definido abaixo.

Definição 3-10 Operador Posicional (COSTAGLIOLA; CHANG, 1999)

Dada uma gramática posicional linear LPG = (N, T, S, P, POS, PE) e uma relação REL ∈ POS, então para todo x, y ∈ N ∪ T de modo que "x REL y" ocorra do lado direito de uma regra de produção na LPG, o operador posicional REL correspondente é definido como:

$$\text{REL}(i) = j \text{ if } w[i] \in \text{LAST}(x) \text{ and } w[j] \in \text{FIRST}(y)$$

Onde LAST(x) é o conjunto de terminais que podem aparecer no final de uma sentença posicional derivada de x, FIRST(x) é o conjunto de terminais que podem aparecer no início de uma sentença posicional derivada de y, e w é a entrada. ▪

O formalismo é concluído com a definição de aceitabilidade a seguir.

Definição 3-11 LPG pLR Aceitável (COSTAGLIOLA; CHANG, 1999)

Uma gramática linear posicional (N, T, S, P, POS, PE) é definida como pLR aceitável se e somente se é possível construir o operador posicional correspondente para cada relação em POS. ▪

Dadas as onze definições, (COSTAGLIOLA; CHANG, 1999) apresentam o funcionamento do analisador de uma gramática posicional segundo os algoritmos abaixo. O Algoritmo 3-1 abaixo descreve o funcionamento do analisador sintático pLR. O Algoritmo 3-2 descreve a construção da tabela SpLR(1). E o Algoritmo 3-3 trata da conversão de uma gramática SpLR(1) em uma SLR(1).

Algoritmo 3-1 Analisador pLR (COSTAGLIOLA; CHANG, 1999)

Entrada: Uma figura p (representada por um vetor de *tokens* w, um índice inicial em w e uma lista Q de pares (pos, i)), uma especificação de um conjunto operadores posicionais e uma tabela de análise pLR para uma gramática linear posicional LPG pLR aceitável, como definido acima.

Saída: se p está em L(LPG), uma árvore de análise *bottom-up*; caso contrário uma indicação de erro.

Método: Inicialmente, o analisador tem s_0 em sua pilha, onde s_0 é o estado inicial, e w em seu *buffer* de entrada. O analisador executa o programa a seguir, até que a ação "aceitar" ou "erro" seja encontrada.

```
início
    seja la o índice inicial retornado por SP();
    repita para sempre
    início
        seja s o estado no topo da pilha e a o símbolo indexado por la;
        se action[s, a] = shift s′ então
        início
            insira a depois s′ no topo da pilha;
            seja ip = la marque w[ip];
            seja OP o operador em position[s′];
            seja la = OP(ip) de modo que w[la] não seja marcado;
        fim
        senão se action[s, a] = reduce A → x então
        início
            remova 2*|x| símbolos da pilha;
            seja s′ o estado agora no topo da pilha;
            insira A depois goto[s′, A] no topo da pilha;
            escreva a produção A → x na saída do algoritmo;
        fim
        senão se action[s, a] = accept então
```

retorne

senão erro()

fim;

fim.

Note que a marcação dos símbolos não permite que o mesmo símbolo seja considerado mais de uma vez. ▪

Este algoritmo é uma simples extensão do algoritmo 4.7 de (AHO; AHO, 2007), no qual a principal diferença é o preenchimento da coluna 'Position', descrito a seguir.

Algoritmo 3-2 Construção da tabela SpLR(1) (COSTAGLIOLA; CHANG, 1999)

Entrada: Uma pLR gramática posicional linear aumentada LPG′;

Saída: A tabela SpLR(1) contendo as seções Action, GoTo e Position para LPG′.

Método:

1. Construa C = {I_0, I_1, ..., I_n}, a coleção de conjuntos de itens de pLR(0) para LPG′.
2. Estado i é construído de I_i. As seções Action e Position para o estado i são determinadas como se segue:
   a) Sempre que A → [α • REL β] pertencer a I_i, adicione REL em position[I_i].
   b) Se [A → α • REL a β] pertencer a I_i e GoTo(I_i, a) = I_j , então preencha Action[i, a] com "shift j" ("s_j"). Onde 'a' deve ser um símbolo terminal. Note que REL é ignorada.
   c) Se [A → α •] pertencer a I_i, então preencha Action[i, a] com "reduce A → α" para todo 'a' em FOLLOW(A) e adicione POSFOLLOW(A) em Position[I_i]. Onde A não deve ser S′.
   d) Se [S′ → SP S •] estiver em I_i, então preencha Action[i, $] com "accept" e adicione ANY em Position[i].
   Caso qualquer conflito seja gerado em Action ou Position(excetuando-se ANY) pelas regras acima, se diz que a gramática não é SpLR(1). O algoritmo falha em produzir um analisador, neste caso.
3. As transições em GoTo para o estado i são construídas para todos os não-terminais A usando a regra: Se GoTo(I_i , A) = I_j, então GoTo[i, A] = j.
4. Todas as entradas não definidas pelas regras (2) e (3) são preenchidas com "error".
5. O estado inicial do analisador é o construído com [S′ → • SP S]. ▪

Por fim, (COSTAGLIOLA; CHANG, 1999) apresentam como converter uma SpLR(1) em SLR(1), permitindo seu uso em geradores de compiladores como o YACC.

Algoritmo 3-3 Conversão de uma gramática SpLR(1) em uma SLR(1) com ações (COSTAGLIOLA; CHANG, 1999)

Entrada: Uma gramática SpLR(1) de LPG = (N, T, S, P, POS, PE) e as implementações dos operadores correspondentes as relações em POS.

Saída: Uma gramática SLR(1) com ações G = (N′, T′, S′, P′), de modo que p ∈ L(LPG) se e somente se (w, Q, SP) é aceito por uma implementação YACC de G. Onde (w, Q, SP) é a representação da entrada de p como definido neste capítulo.

Método:

seja N′=N, T′=T, S′=S, P′=Ø
para cada produção p em P faça
início
    seja p′ a produção obtida pela remoção de todos as relações espaciais em p
    se p é do tipo A → α a REL β então
        substitua 'a' por "a REL" em p′ e adicione REL em N′
    se p é do tipo A → α a então
    início
        se POSFOLLOW(A) = {REL} então
            substitua 'a' por "a REL" em p′ e adicione REL em N′
        se POSFOLLOW(A) = {REL, ANY} então
            substitua 'a' por "a REL_ANY" em p′ e adicione REL_ANY em N′
    fim
    adicione p′ em P′
fim
para cada REL (ou REL_ANY, respectivamente) adicionado em N′ na instrução "para cada" anterior faça
    adicione REL → {REL()} (ou REL_ANY → {REL_ANY()}, respectivamente) em P′
fim. ▪

Este algoritmo é apenas parte do formalismo. É perfeitamente possível conceber uma gramática diretamente no formato de saída do Algoritmo 3-3. Se ele for aplicado à gramática da escada apresentada acima, resultará na seguinte gramática SLR(1) com ações:

S → **SP** STAIRS
STAIRS → STEP **BEFORE** STAIRS
STAIRS → STEP
STEP → *stringer* **UNDER** *tread*
BEFORE → {`BEFORE()`}
UNDER → {`UNDER()`}

Como pôde ser visto, (COSTAGLIOLA; CHANG, 1999) apresentam uma técnica capaz de analisar e aceitar figuras bidimensionais a partir de analisadores gramaticais LR. Assume-se que a mesma técnica possa ser utilizada em espaço de *voxels* para a interpretação de estruturas em jogos. Outros trabalhos relacionados a gramáticas posicionais livres de contexto podem ser encontrados em (COSTAGLIOLA et al., 1993, 1998, 2003; COSTAGLIOLA; CHANG, 1990; COSTAGLIOLA; DEUFEMIA; POLESE, 2007; COSTAGLIOLA; POLESE, 2000; COSTAGLIOLA; TOMITA; CHANG, 1991).

Capítulo 4 - Contextualização

A fim de ambientar o leitor no contexto em que essa dissertação atua, este capítulo dedica-se a detalhar características relevantes do jogo e das estruturas nele contidas. Além disso, são discutidas questões relativas à criação de bots de Minecraft através das tecnologias disponíveis até o momento.

4.1 O jogo

Os principais conceitos do jogo relevantes para este trabalho são: blocos e personagens. Blocos são implementações de *voxels*, representados como cubos unitários que possuem uma variedade de atributos, incluindo, por exemplo, tipo, rigidez, fluidez e luminosidade. Personagens dividem-se em duas categorias: jogadores (humanos ou *bots*) e não jogadores (animais, monstros e habitantes de vilarejos). O espaço do Minecraft pode ser descrito como sendo um espaço euclidiano de três dimensões (BERGER; COLE, 1987). Este espaço é regulado por dois tipos diferentes de 'grades': uma possui valores inteiros e se aplica somente aos blocos e a outra de valores reais que se aplica às personagens habitantes do jogo. Isso significa que blocos não podem ocupar posições com valor real, pois estariam infringindo a grade de valores inteiros. Em oposição a isso, personagens podem se mover mais suavemente sobre o espaço, pois seu espaço é definido por valores reais.

O terreno é gerado de forma procedural (HENDRIKX et al., 2013) e sob demanda, ou seja, conforme os jogadores navegam pelo terreno, as extremidades vão sendo estendidas pela geração de novas formações geográficas. Após a geração do relevo, este é preenchido com certas estruturas naturais, como árvores e cactos, e construções, como casas e torres. Os algoritmos que geram tais estruturas são específicos para cada classe de estrutura. Eles em geral utilizam geradores de números aleatórios para definir os tamanhos e posicionamento de partes de estruturas. (SPORTELLI; TOTO; VESSIO, 2014) propõe uso de gramáticas probabilísticas para essa função. A geração dessas estruturas tem como produto um conjunto de blocos posicionado adequadamente segundo a classe da estrutura gerada. Nenhuma outra informação é mantida envolvendo tais blocos

como uma entidade. Fica a cargo dos jogadores o reconhecimento de cada estrutura como pertencente a uma classe, e isso é feito de forma puramente visual por jogadores humanos.

Em geral, os blocos podem ser destruídos ou construídos por jogadores a fim de modificar o espaço ao seu redor. No Minecraft existem pelo menos dois modo de jogo: criativo e sobrevivência. No modo sobrevivência, o jogador deve vencer os desafios naturais, como fome, frio, queda de grandes alturas, afogamento e criaturas selvagens inimigas. Comumente, isso deve ser feito enquanto ele minera seus recursos e fabrica artesanalmente seus utensílios e construções. No modo criativo o jogador possui a liberdade de alterar o espaço tendo todo o repertório de blocos e utensílios à sua disposição, além de não ter de se preocupar com ameaças, graças à sua imortalidade nesse modo de jogo.

Sua arquitetura cliente-servidor permite que as diferentes aplicações sejam responsáveis por partes distintas da constituição do jogo. A aplicação servidora é responsável, entre outras coisas, por gerar o terreno em seus biomas naturais e centralizar as mudanças feitas pelos jogadores na disposição dos blocos. As aplicações cliente são responsáveis por '*renderizar*' os blocos e exibir a visão de cada personagem a seus respectivos jogadores, além de receber destes os comandos a serem enviados ao servidor. A comunicação entre cliente e servidor se dá por meio de um protocolo próprio. Os dados transitados são limitados às informações de interesse de cada aplicação cliente de acordo com a personagem em questão. Essas informações incluem os dados dos blocos e personagens próximas, enviadas do servidor para o cliente enquanto este envia as movimentações e ações do jogador sobre os elementos aos quais tem acesso. Vale reforçar que, após a geração de uma estrutura, nenhum rastro capaz de representar tal estrutura é mantido entre os blocos. Os dados de blocos são constituídos única e exclusivamente pelos valores independentes de cada bloco em cada atributo.

O jogo possui grande flexibilidade para sofrer modificações, os chamados *mods*. A intensa adoção da comunidade de jogadores, em união com esta flexibilidade, deu origem a uma grande variedade de versões do jogo sendo praticadas em diversos servidores. Em muitos casos, os administradores desses servidores projetam regras internas que mudam os objetivos dos jogadores e, na prática, criam um 'subjogo', usando o Minecraft como uma plataforma pra criação desses 'sabores' de jogo. Este fato expande amplamente o espaço das possibilidades para os *bots*, pois eles podem ser projetados segundo o propósito específico de atuar em uma versão modificada do jogo, e isto inclui reconhecer estruturas de classes praticadas apenas em certos *mods*.

4.2 As estruturas

Existem duas origens possíveis de uma dada estrutura no Minecraft: ela pode ter sido gerada pelo jogo para compor o cenário ou pode ser obra de algum jogador. Em ambos os casos, é de interesse dos jogadores poder identificá-las. Entre as estruturas originais de maior destaque estão as construções dos vilarejos, como casas, poços d'água, torres, postes e plantações. Os vilarejos são habitados por seres humanoides não jogadores com os quais os jogadores podem interagir e negociar mercadorias. Encontrar um vilarejo indica a possível presença desses habitantes. Já uma estrutura não originalmente criada pelo jogo indica a passagem de um jogador por aquela localidade. Isso pode significar a existência de recursos naturais interessantes nas proximidades e até de itens de valor abandonados ou escondidos por quem passou por ali anteriormente. Vale ainda mencionar que a mecânica flexível do jogo permite a criação de uma infinidade de construções, e esse aspecto tem sido particularmente explorado pela comunidade de jogadores do Minecraft. Entre as construções mais grandiosas encontram-se réplicas de cidades e monumentos, como nas figuras 4-1 e 4-2 abaixo.

Figura 4-1 Uma cidade no Minecraft

Figura 4-2 Réplica do Coliseu no Minecraft

4.3 Os bots

No Minecraft, a criação de um *bot* deve respeitar a arquitetura cliente-servidor. Para que um *bot* atue no jogo no controle de uma personagem (assim como um jogador humano), basta que este aja como aplicação cliente e atenda ao protocolo de comunicação com servidor. Estes *bots* terão acesso às localizações de blocos em suas proximidades e poderão atuar sobre o espaço enviando os comandos adequados ao servidor. Atualmente os *bots* existentes limitam-se a receber comandos de outros jogadores por meio de mensagens de *chat*, como em (BEAUMONT, 2015; KUBALL, 2013b). A arquitetura geral dos *bots* atuais segue o que é ilustrado na Figura 4-3 abaixo. Sobre uma camada de tratamento do protocolo Minecraft opera um módulo de tratamento de mensagens de *chat* que é responsável por acionar funções num segundo módulo de tarefas. A execução de cada tarefa pode envolver acesso de leitura e escrita na camada de protocolo, incluindo mensagens *chat*. Assim o *bot* é capaz de receber comandos, executá-los segundo as tarefas que este suporta e comunicar-se com os demais jogadores.

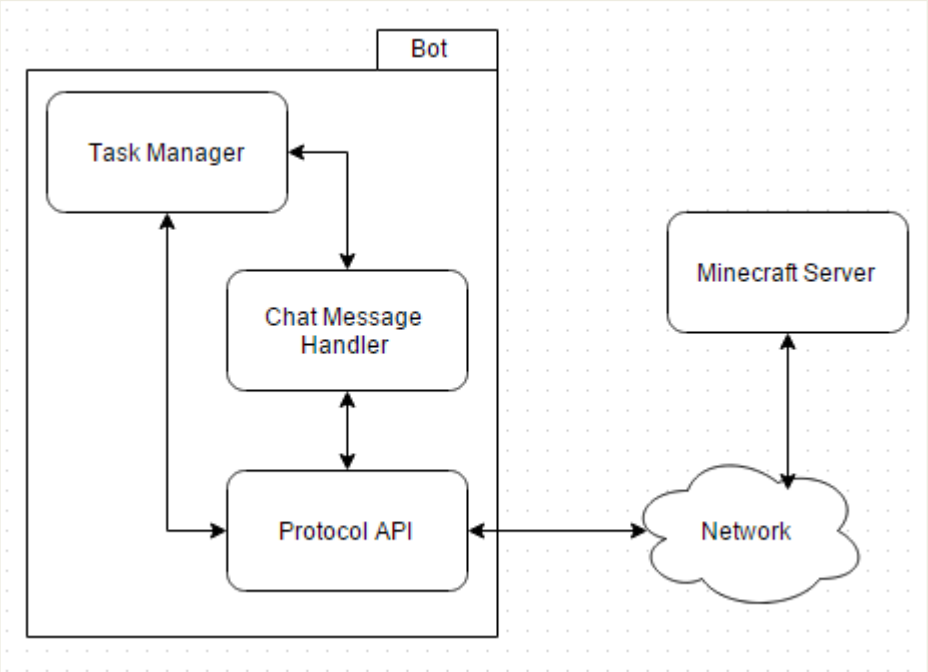

Figura 4-3 Estrutura geral de um bot de Minecraft

O conjunto das tarefas de *bots* complexas e desejáveis por jogadores humanos inclui, por exemplo:

- **Mineração:** encontrar e coletar certos materiais pelo terreno, o que envolve avistar uma região de interesse e locomover-se até ela, mesmo que isso envolva escavar túneis e construir pontes ou escadas;
- **Colheita:** uma extensão da tarefa de mineração, que envolve ainda preparar uma próxima colheita, plantando os vegetais e alimentando os animais;
- **Vigilância:** circular por uma região à procura de ameaças e combatê-las conforme necessário;
- **Batalha:** ativamente engajar-se em confronto com inimigo designado, com a finalidade de conquista de terreno ou recursos;
- **Construção**: dispor ou remover blocos de modo a reservar o espaço de um aposento, passagem ou salões.

Se consideradas as versões modificadas, o uso de *bots* inteligentes permitiria testar e validar cenários de um ‘subjogo’ interno criado no Minecraft. Esse uso permite incrementar a qualidade e, consequentemente, valorizar economicamente uma dada versão modificada. Além disso, é possível utilizar o Minecraft como plataforma para

pesquisa científica, como demostrado por (ABEL et al., 2015; ABEL; TELLEX, 2015; BARTH-MARON et al., 2014) no desenvolvimento, teste e aprimoramento de algoritmos de priorização de ações para robôs físicos (STACEY, 2015).

Capítulo 5 - Proposta

Este capítulo apresenta um aprofundamento do problema abordado e a descrição de relações posicionais básicas, a definição de diferença posicional e o algoritmo do analisador léxico espacial para viabilização da leitura de espaços multidimensionais de entrada.

Após pesquisa e investigação das técnicas existentes com potencial de oferecer uma solução ao problema, as gramáticas posicionais livres de contexto apresentaram-se tanto como um formalismo teórico adequado quanto como uma vantagem prática, a redução para gramáticas SLR. A família de gramáticas LR – que inclui SLR, LALR e GLR – tem larga adoção na indústria e dispõe de uma gama de implementações em ferramentas para criação de compiladores-de-compiladores (BONE, 2014).

Segundo (AHO; AHO, 2007), a interação entre o analisador léxico e o sintático se dá com o analisador sintático solicitando ao léxico o próximo símbolo lido da entrada. Em, seguida, já de posse deste símbolo, o analisador sintático segue seu funcionamento e pode solicitar mais um símbolo ao analisador léxico ou decidir entre aceitar e rejeitar a entrada. Assim, tem-se que o analisador léxico precisa decidir qual novo símbolo deve ser passado para o analisador sintático sem ter o conhecimento do estado da análise sintática, ou seja, fazendo uso somente dos valores de entrada e da posição atual da leitura.

Para efetuar a interpretação de um corpo textual gerado a partir de uma gramática, é preciso decidir a direção de leitura e a direção de redução das regras gramaticais. No caso dos analisadores LR e LL a leitura é feita da esquerda para a direita. Entretanto, os LR procedem as reduções a partir da extremidade mais à direita para a esquerda, enquanto que os LL o fazem a partir da ponta mais à esquerda para a direita. O trabalho de um analisador léxico operando dessa forma em exatamente uma dimensão é ler um caractere por vez da esquerda para a direita em busca de símbolos definidos em sua configuração. Na ocasião de encontrar um símbolo, este deve ser passado ao analisador sintático para que se proceda a análise.

Quando a dimensionalidade é passada para o plano bidimensional, ou seja, dois graus de liberdade, surge uma problema no quesito direção de leitura. Note que, apesar de textos serem normalmente exibidos em duas dimensões, eles são, na verdade,

unidimensionais. E lembre-se de que as linhas são na verdade trechos de uma única sequência contínua e linear de caracteres – como numa fita –, na qual a quebra de linha é apenas mais um deles. Pela presença de mais um grau de liberdade, o caminho de leitura descrito por uma direção única, como nos casos LR e LL, não é capaz de alcançar todos os pontos na entrada ou, nesse caso, plano de entrada.

Quanto maior a dimensionalidade, mais graus de liberdade e, portanto, mais complexo se torna efetuar uma leitura léxica pelo espaço. As gramáticas posicionais livres de contexto de (COSTAGLIOLA; CHANG, 1999) fornecem uma forma eficaz de realizar essa leitura. As relações posicionais entre os símbolos do lado direito das regras de produção têm o papel de guiar a leitura da entrada por meio da coluna 'Position' adicionada à tabela SLR e transformando-a na tabela SpLR. Entretanto, ao converter a gramática de volta para SLR – e aplicar as ações dos operadores posicionais às regras de produção dos símbolos de relações posicionais – a informação da coluna 'Position' é perdida, simplesmente por não existir numa tabela SLR. E, em contrapartida, são adicionadas ações às regras de produção relativas às relações posicionais.

Como, então, poderia um analisador léxico fornecer símbolos para o analisador sintático se não é possível que uma direção linear de leitura abranja um espaço multidimensional? Em uma dimensão, ler o próximo caractere em uma direção predefinida é suficiente para cobrir a entrada. Em mais dimensões, porém, não é esse o caso. A redução da regra de produção que acionaria a ação de um operador posicional só será executada após a leitura do símbolo seguinte. Há um problema de ordem dos eventos.

De modo a permitir que a leitura da entrada seja feita segundo o descrito pela gramática, é proposta desta dissertação o analisador léxico espacial, descrito abaixo. Este analisador léxico recebe como entrada a gramática posicional e as relações posicionais. A varredura do espaço é feita segundo as relações posicionais entre os símbolos da gramática. Além disso, são definidas abaixo relações posicionais básicas. Estas, por sua simplicidade, são capazes de contribuir na definição de gramáticas posicionais livres de contexto que descrevem uma variedade de classes de estruturas, entre estes alguns dos mais comumente encontrados em cenários naturais do Minecraft.

5.1 As Relações Posicionais

O papel das relações posicionais é guiar o caminho de leitura feito pelo analisador léxico. O conjunto das relações posicionais possíveis é dado por todas as combinações

possíveis de posições relativas a uma dada origem, onde cada combinação é uma relação posicional. Esse conjunto é da ordem de 2^s, onde *s* é o tamanho do espaço, o que no caso tridimensional é o volume, dado pelo produto x * y * z dos tamanhos do espaço em cada dimensão. Se o espaço for infinito em qualquer dimensão, então o conjunto de relações posicionais também será. Entretanto, algumas dessas relações posicionais são mais valiosas do que outras para geração de estruturas comumente encontradas e praticadas no espaço do jogo.

Nas definições de relações posicionais desta dissertação, os blocos escuros (em azul) representam a posição atual de leitura (ou *p1*), os blocos claros (em laranja) representam as novas posições possíveis segundo cada relação posicional (ou *p2*), e a orientação dos eixos coordenados é como indicado na Figura 5-1 abaixo, com a coordenada *y* representado a vertical, enquanto o plano horizontal é dado por *xz*:

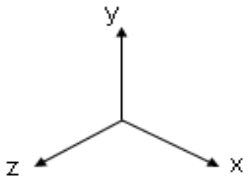

Figura 5-1 Orientação dos eixos coordenados

Uma forma de representar as relações posicionais diferentemente da apresentada por (COSTAGLIOLA; CHANG, 1999) é pelo conjunto de vetores *n*-dimensionais das diferenças entre cada posição final e a inicial. Esses vetores devem ser somados a uma dada posição inicial para que assim sejam obtidas as posições finais definidas pela relação posicional. Segue abaixo a definição formal.

Definição 5-1 Diferença posicional (δ)

Sejam POS o conjunto finito de identificadores de relações posicionais, P o conjunto de posições possíveis. O conjunto de diferenças posicionais de uma relação posicional é o mapeamento δ: POS → P. De modo que:

$$\delta(REL) = \{ location(b) - location(a) \mid b \in REL(a) \}$$

Onde REL ∈ POS é uma relação posicional, *a* é uma posição inicial e *b* é uma posição final, ambas as posições segundo REL. ▪

Esta notação simplificará as definições de relações posicionais nesta dissertação e pode ser usada pelo analisador léxico para obter as posições produzidas pelo operador posicional de uma relação. Abaixo são listadas as relações posicionais básicas utilizadas nas gramáticas do próximo capítulo. Em alguns casos, essas relações posicionais serão expandidas ou combinadas para formar outras relações espaciais. Isso será feito conforme necessário ao entendimento das gramáticas em que serão utilizadas.

5.1.1 ABOVE

Uma posição *p1* será considerada acima de (do inglês, *above*) outra posição *p2* se e somente se location(*p1*) = (x, y, z) e location(*p2*) = (x, y-1, z), ou seja, se δ(ABOVE) = {(0,-1,0)}; como ilustrado na Figura 5-2 abaixo:

Figura 5-2 ABOVE

5.1.2 UNDER

Uma posição *p1* será considerada debaixo de (do inglês, *under*) outra posição *p2* se e somente se location(*p1*) = (x, y, z) e location(*p2*) = (x, y+1, z), ou seja, se δ(UNDER) = {(0,1,0)}; como ilustrado na Figura 5-3 abaixo:

Figura 5-3 UNDER

5.1.3 FACING

Uma posição *p1* será considerada de frente para (do inglês, *facing*) outra posição *p2* se e somente se location(*p1*) = (x, y, z) e location(*p2*) ∈ {(x, y, z+1), (x, y, z-1)}, ou seja, se δ(FACING) = {(0,0,1), (0,0,-1)}; como ilustrado na Figura 5-4 abaixo:

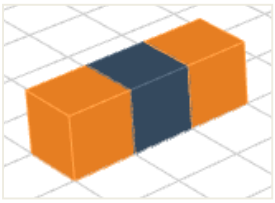

Figura 5-4 FACING

5.1.4 BESIDE

Uma posição *p1* será considerada ao lado de (do inglês, *beside*) outra posição *p2* se e somente se location(*p1*) = (x, y, z) e location(*p2*) ∈ {(x+1, y, z), (x-1, y, z)}, ou seja, se δ(BESIDE) = {(1,0,0), (-1,0,0)}; como ilustrado na Figura 5-5 abaixo:

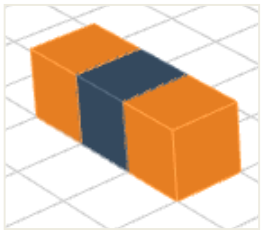

Figura 5-5 BESIDE

5.1.5 AROUND

Uma posição *p1* será considerada ao redor de (do inglês, *around*) outra posição *p2* se e somente se location(*p1*) = (x, y, z) e location(*p2*) = (k, y, s), onde k ∈ [x-1, x+1] e s ∈ [z-1, z+1], ou seja, se δ(AROUND) = {(0,0,1), (-1,0,0), (0,0,-1), (1,0,0), (1,0,1), (-1,0,1), (-1,0,-1), (1,0,-1)}; como ilustrado na Figura 5-6 abaixo:

Figura 5-6 AROUND

5.2 O analisador léxico espacial

O analisador léxico espacial começa a leitura a partir da posição inicial, definida pelo espaço de entrada fornecido. Tomando como base a posição atual, o analisador

decide a posição seguinte consultando as relações posicionais fornecidas. Cada relação posicional fornece um conjunto de posições candidatas através de seu operador posicional respectivo. O analisador léxico espacial verifica cada posição candidata em busca de um símbolo posicionado adequadamente para que o analisador léxico possa seguir suas reduções e empilhamentos. Ao encontrar uma nova posição viável, o analisador avança sobre ela e repassa seu conteúdo ao analisador sintático, e este dará prosseguimento à análise. De modo a viabilizar esse comportamento, a seguinte heurística foi adotada:

> *A próxima posição de leitura será aquela que contém um símbolo aceitável como próximo caso este esteja localizado em posição aceitável segundo uma relação posicional capaz de relacionar adequadamente as posições de ambos estes símbolos.*

No Algoritmo 5-1 abaixo é apresentado o funcionamento do analisador léxico posicional, fazendo uso da heurística acima e dos operadores posicionais, a partir de sua posição atual, para obter novas posições candidatas e verificar a viabilidade dessas posições candidatas segundo o descrito na Definição 3-10.

Algoritmo 5-1 Analisador léxico espacial

Entrada: A posição atual p_0, o símbolo s_0 desta posição, o conjunto P de regras de produção, o conjunto POS de relações posicionais e seus operadores

Saída: A próxima posição de leitura, ou vazio, indicando fim da leitura

Método:

para cada regra de produção *r* em P
início
    para cada relação posicional REL em *r*
    início
        sejam os símbolos α e β de modo que [α REL β] é um trecho de *r*;
        se $s_0 \in$ LAST(α) então
            seja OP o operador posicional correspondente a REL;
            para cada posição candidata p_c em OP(p_0)
            início
                seja s_c o símbolo candidato em p_c;
                se $s_c \in$ FIRST(β) então

retorne p_c;

fim

fim

fim.

Dessa forma, a cada passo, algumas posições serão sondadas pelo analisador léxico. Essas posições representam um conjunto muito menor do que o de todas as posições possíveis para busca da nova posição de leitura. Isso se dá pelo fato de a leitura ser guiada pelas relações posicionais e seus operadores. A complexidade desse algoritmo (BACHMANN, 1894; KNUTH, 1976; LANDAU, 1909) é dada por:

$$\Theta(n\ r\ p\ \Theta(a))$$

Onde:

- n é o número de posições candidatas dadas por uma relação posicional;
- r é o número de relações posicionais;
- p é o número de posições em um objeto de entrada; e
- $\Theta(a)$ é a complexidade de acessar uma posição na entrada.

5.3 Exemplo

Considere, por exemplo, um poste de luz do Minecraft como ilustrado na Figura 5-7. Ele é formado por um conjunto de blocos com uma sequência vertical ininterrupta de blocos tipo *fence* (Figura 5-8) seguida por um bloco *cloth* (Figura 5-9) e contendo um bloco *torch* (Figura 5-10) horizontalmente ao seu redor. É importante ressaltar que os blocos *fence* e *torch*, apesar de não serem representados visualmente por um cubo como o bloco *cloth*, ocupam um espaço cúbico com volume idêntico ao bloco *cloth* e, portanto, são considerados *voxels* sem nenhum prejuízo à sua participação na análise.

Figura 5-7
Poste de luz

Figura 5-8
Bloco *fence*

Figura 5-9
Bloco *cloth*

Figura 5-10
Bloco *torch*

Uma gramática capaz de descrever objetos desse tipo segue abaixo:

LAMPPOST → BODY **UNDER** HEAD
BODY → BODY **UNDER** *fence*
BODY → *fence*
HEAD → *cloth* **AROUND** *torch*

Aplicando-se o Algoritmo 3-3 na gramática acima, obtém-se como resultado a seguinte gramática SLR:

S → **SP** LAMPPOST
LAMPPOST → BODY **UNDER** HEAD
BODY → BODY **UNDER** *fence*
BODY → *fence*
HEAD → *cloth* **AROUND** *torch*
SP → {`SP`()}
UNDER → {`UNDER`()}
AROUND → {`AROUND`()}

Note que nas três últimas regras desta gramática os símbolos não terminais da esquerda são levados a ε (épsilon), ou seja, ao vazio, em termos de símbolos. Este exemplo segue a notação utilizada em (COSTAGLIOLA; CHANG, 1999), com o trecho de ação do lado direito da regra de produção. Nos exemplos seguintes, essa notação omitirá o trecho de ação e exibirá ε (épsilon) como lado direito da regra de produção.

A leitura de um poste como o ilustrado na Figura 5-7 deve ter como posição inicial o bloco tipo *fence* mais inferior na dimensão vertical, e o caminho de leitura sobe verticalmente (no eixo *y*) até o bloco tipo *cloth*, no qual então é feita uma varredura horizontal ao redor deste, a procura de um bloco tipo *torch*, como indicado na Figura 5-11 abaixo:

Figura 5-11 Caminho do analisador léxico espacial sobre o poste de luz

Ou seja, suponha que a base do poste se encontra na origem, o ponto (0, 0, 0), e que esta é a posição inicial fornecida ao analisador deste exemplo. O analisador sintático começa por empilhar o bloco tipo *fence* e em seguida solicita ao analisador léxico o próximo símbolo de leitura. Este por sua vez varre as regras de produção e seleciona a relação posicional UNDER, por ser a única com *fence* ∈ LAST(BODY), já que BODY é o único termo que precede UNDER. Esta relação posicional leva o analisador léxico ao ponto (0, 1, 0), imediatamente acima da posição inicial, onde há um segundo bloco tipo *fence*. Como essa posição fornece um bloco aceitável para as produções da gramática, este novo símbolo é então repassado ao analisador sintático, que, ao recebê-lo efetua as reduções adequadas e volta a solicitar um novo símbolo. Desta vez o processo anterior é repetido, e um bloco tipo *cloth* é encontrado na nova posição candidata. Este também é um bloco aceitável pelas regras de produção da gramática e, portanto, repassado ao analisador sintático. Este bloco causa um empilhamento de estado no analisador sintático, que novamente solicita um novo símbolo. Agora o analisador léxico espacial tem o bloco *cloth* no ponto (0, 2, 0) como posição atual de leitura e, analogamente às etapas anteriores, seleciona o operador posicional AROUND para investigar uma nova posição. Este operador contorna a posição corrente em busca de um bloco tipo *torch* e, ao encontrá-lo em (1, 2, 0), faz com que o símbolo seja passado do analisador léxico para o sintático, que conclui a análise e decide por aceitar a entrada.

Note que, o analisador léxico espacial não depende de nenhuma característica da entrada, nem de sua dimensionalidade, nem de seu sistema de coordenadas, nem de qualquer outra informação. De fato, se for estabelecido um protocolo ou interface de comunicação com uma dada implementação do analisador léxico espacial, este é capaz de operar sobre qualquer espaço de entrada. Isso é possível pois o operador léxico espacial funciona como um agente conciliador entre a gramática, as relações posicionais e o

espaço de entrada. Assim sendo, o analisador léxico espacial e, consequentemente, as gramáticas posicionais livres de contexto são capazes de atuar em, por exemplo, espaços contínuos ao invés de discretos, ou em coordenadas polares. Para isto, basta que as relações posicionais atuem adequadamente sobre o espaço em que operam. É preciso notar, entretanto, que, mesmo em espaço contínuos, a leitura de símbolos ainda deve ocorrer de forma discreta, com um símbolo em um posição por vez.

Capítulo 6 - Gramáticas

Neste capítulo são apresentados exemplos de classes de objetos comumente encontrados no jogo. Cada classe objeto será acompanhada de sua gramática posicional e suas relações posicionais associadas.

O ambiente natural do Minecraft é repleto de objetos gerados automaticamente com a finalidade de complementar a paisagem e oferecer um cenário natural aos seus jogadores. Além de formações naturais, como diferentes tipos de árvores, cactos e cachoeiras, e estruturas artificiais, como casas, torres, plantações, poços d'água e celeiros, o jogo permite que os jogadores criem estruturas livremente pelo espaço. Abaixo estão listados alguns exemplos de classes de objetos tratados interpretáveis por gramáticas posicionais livres de contexto. Esses objetos são comumente encontrados no Minecraft. As gramáticas posicionais são equivalentes às obtidas após conversão pelo Algoritmo 3-3.

6.1 Árvore

A concepção de uma árvore (Figura 6-1), por ser um objeto relativamente simples, requer, nos termos deste trabalho, apenas dois tipos de bloco: *log* (Figura 6-2) e *leaves* (Figura 6-3). Um conjunto de blocos precisa constituir um caule sob uma copa, para ser considerado árvore nos termos deste trabalho. Um caule é uma sequência vertical ininterrupta de blocos tipo *log* de tamanho variável, e uma copa é um aglomerado de blocos tipo *leaves*.

Figura 6-1 Árvore

Figura 6-2
Bloco *log*

Figura 6-3
Bloco leaves

Para prosseguir a descrição da árvore é preciso expandir e unir os conceitos de AROUND e UNDER para formar a relação abaixo. Na Figura 6-4 da relação posicional AROUND_UP, o bloco da posição inicial (em azul-escuro) encontra-se oculto pelas numerosas posições candidatas (em laranja-claro). Ele está localizado no centro da camada inferior da figura.

6.1.1 AROUND_UP

Uma posição *p1* será considerada ao redor de, ou abaixo de (do inglês, *around and up*) outra posição *p2* se e somente se location(*p1*) = (x, y, z) e location(*p2*) = (k, u, s), onde k ∈ [x-2, x+2], u ∈ [y, y+1], e s ∈ [z-2, z+2], como ilustrado na Figura 6-4 abaixo:

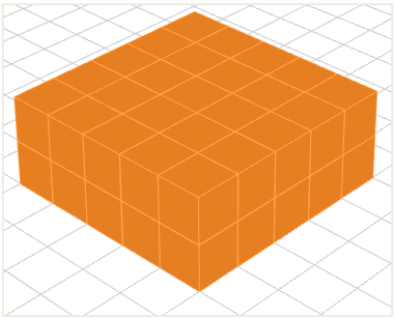

Figura 6-4 AROUND_UP

6.1.2 Gramática

Segue abaixo a gramática da árvore:

S → **SP** TREE
TREE → STALK **AROUND_UP** TREETOP
STALK → STALK **UNDER** *log*

STALK → *log*

TREETOP → TREETOP **AROUND_UP** *leaves*

TREETOP → *leaves*

SP → ε

UNDER → ε

AROUND_UP → ε

Esta gramática exige uma leitura de baixo para cima na árvore, e a posição inicial esperada é, portanto, a base da mesma. Iniciar a leitura em outra posição resultará na rejeição da entrada, caso essa posição não pertença ao caule da árvoreou na leitura parcial dos blocos, caso pertença. A geração dos estados e a tabela do analisador encontram-se a seguir:

I0: S → • **SP** TREE

TREE → • STALK **AROUND_UP** TREETOP

STALK → • STALK **UNDER** *log*

STALK → • *log*

I1: S → **SP** TREE •

I2: TREE → STALK • **AROUND_UP** TREETOP

STALK → STALK • **UNDER** *log*

TREETOP → • TREETOP **AROUND_UP** *leaves*

TREETOP → • *leaves*

I3: STALK → *log* •

I4: TREE → STALK **AROUND_UP** TREETOP •

TREETOP → TREETOP • **AROUND_UP** *leaves*

I5: STALK → STALK **UNDER** *log* •

I6: TREETOP → *leaves* •

I7: TREETOP → TREETOP **AROUND_UP** *leaves* •

Tabela 6-1 Tabela LR estendida (com coluna *Position*) da gramática da árvore

State	Action			GoTo			Position
	$	*leaves*	*log*	TREE	STALK	TREETOP	
0			S3	1	2		SP

1	acc						ANY
2		S6	S5			4	AROUND_UP, UNDER
3		R3	R3				AROUND_UP, UNDER
4	R1	S7					AROUND_UP
5		R2	R2				AROUND_UP, UNDER
6		R5					AROUND_UP
7		R4					AROUND_UP

Como pôde ser visto na Tabela 6-1, existem conflitos na coluna Position; segundo o Algoritmo 3-2, esta gramática não é SpLR(1), e um analisador não poderia ser produzido para ela. Entretanto, como as relações posicionais AROUND_UP e UNDER possuem intercessão entre as posições candidatas, e dado o modo de funcionamento do analisador léxico, um analisador pode ser perfeitamente construído para reconhecer árvores nestes termos.

Vale ressaltar ainda que as relações posicionais pertencem sempre ao escopo de uma gramática posicional e não há a obrigação de que uma relação posicional definida para uma gramática tenha o mesmo significado que para as demais, ainda que definida sob o mesmo nome.

6.2 Plantação

Existem diversas formas de plantação possíveis no Minecraft. Uma das mais simples e frequentemente encontradas é formada por fileiras de terra plantada em contato com uma fileira de água e contornadas por blocos de madeira. Assim, uma plantação simples (Figura 6-5) foi definida como uma sequência de ‘linhas de plantação’ precedidas e sucedidas de ‘linhas de contorno’. Ambos os tipos de linhas são formados por sete blocos na horizontal. Uma ‘linha de contorno’ é formada apenas por blocos tipo *log*. Uma ‘linha de plantação’ é formada pela sequência: um *log*, dois *farmland* (Figura 6-6), um *water* (Figura 6-7), dois *farmland* e um *log*. Sobre cada bloco *farmland* deve haver ainda um bloco tipo *wheat* (Figura 6-6), que representa as plantas em si. Note que a *wheat* e *water* cabe a mesma observação quanto ao volume mencionado para *fence* e *torch* no caso do poste de luz: não são representadas visualmente como cubos unitários, mas, na prática, ocupam este espaço.

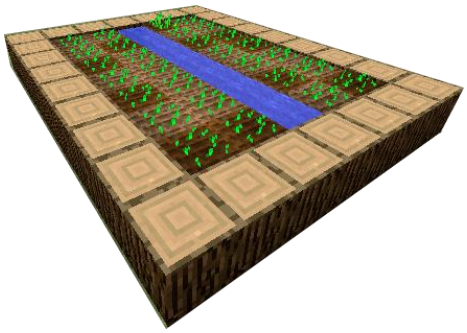

Figura 6-5 Plantação

Figura 6-6 Blocos *farmland* e *wheat*

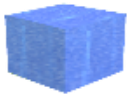

Figura 6-7 Bloco *water*

6.2.1 BEHIND

Uma posição *p1* será considerada atrás de (do inglês, *behind*) outra posição *p2* se e somente se location(*p1*) = (x, y, z) e location(*p2*) = (x, y, z+1), ou seja, se δ(BEHIND) = {(0,0,1)}; como ilustrado na Figura 6-8 abaixo:

Figura 6-8 BEHIND

6.2.2 ASIDE

Uma posição *p1* será considerada atrás de (do inglês, *aside*) outra posição *p2* se e somente se location(*p1*) = (x, y, z) e location(*p2*) = (x, y-1, z+1), ou seja, se δ(ASIDE) = {(0,-1,1)}; como ilustrado na Figura 6-9 abaixo:

Figura 6-9 ASIDE

6.2.3 FOLLOWED_BY

Uma posição *p1* será considerada seguida de (do inglês, *followed by*) outra posição *p2* se e somente se location(*p1*) = (x, y, z) e location(*p2*) = (x+1, y, z-6), ou seja, se δ(FOLLOWED_BY) = {(1,0,-6)}; como ilustrado na Figura 6-10 abaixo:

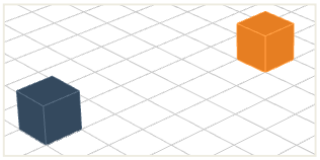

Figura 6-10 FOLLOWED_BY

6.2.4 Gramática

A gramática da plantação:

S → **SP** PLANTATION

PLANTATION → ENDING **FOLLOWED_BY** FILLING **FOLLOWED_BY** ENDING

ENDING → *log* **BEHIND** *log* **BEHIND** *log* **BEHIND** *log* **BEHIND** *log* **BEHIND** *log*

FILLING → FILLING **FOLLOWED_BY** ROW

FILLING → ROW

ROW → *log* **BEHIND** PLANTS **ASIDE** IRRIGATION **BEHIND** PLANTS **ASIDE** *log*

PLANTS → PLANT **ASIDE** PLANT

PLANT → *farmland* **UNDER** *wheat*

IRRIGATION → *water*

SP → ε

FOLLOWED_BY → ε

BESIDE → ε

ASIDE → ε

UNDER → ε

BEHIND → ε

Nota-se, então, que o caminho de leitura do analisador desta gramática posicional efetua uma varredura linha a linha da entrada. Ao final da leitura de cada linha, o analisador passa para a próxima linha executando a relação posicional FOLLOWED_BY. A primeira e última linha são lidas sem variação na componente *y*. As demais têm uma forma serrilhada pelos blocos intermediários, excetuando-se os blocos *log* e *water*, como ilustrado na Figura 6-11 abaixo.

Figura 6-11 Caminho do analisador léxico espacial sobre uma linha de plantação

Note também que esta estrutura é orientada ao longo do eixo *x*. É ainda possível conceber uma plantação orientada ao longo do eixo *z*. Para efetuar a leitura da última, entretanto, é preciso alterar as relações posicionais permutando-se os termos de *x* pelos de *z*.

6.3 Poço d'água

Diferente dos objetos vistos até agora, que podem ser considerados estruturas simples, o poço d'água (Figura 6-12) se destaca por apresentar uma complexidade relativamente maior. Ele é formado por um núcleo de água envolto por paredes de pedra. Estas ainda são rodeadas por um anel de cascalho e cobertas por um telhado de pedra suspenso por colunas de madeira. Em outras palavras, é considerado poço d'água o conjunto de blocos do núcleo de tipo *water* dispostos horizontalmente num formato quadrado dois por dois, seu corpo em contorno ao núcleo em blocos tipo *cobblestone* (Figura 6-13) de altura vertical dois, os suportes de blocos tipo *fence* em cada vértice do superior do corpo, seu teto de blocos tipo *cobblestone* sobre os suportes e, por fim, sua margem de blocos tipo *gravel* (Figura 6-14) de altura 1(um), envolvendo a parte inferior do corpo.

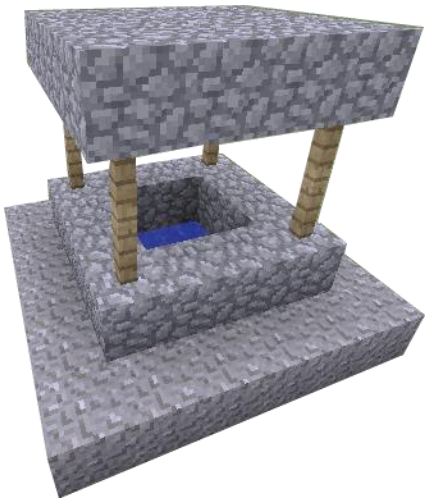

Figura 6-12 Poço d'água

Figura 6-13 Bloco *cobblestone*

Figura 6-14 Bloco *gravel*

Nas figuras das relações posicionais desta e demais seções, os blocos em cinza mais claro servem somente para facilitar a visualização do espaço entre os blocos de interesse (azul-escuro e laranja-claro) e não interferem na definição e funcionamento dos operadores posicionais.

6.3.1 SHALLOW_BESIDE

Uma posição *p1* será considerada ao lado raso de (do inglês, *shallow beside*) outra posição *p2* se e somente se location(*p1*) = (x, y, z) e location(*p2*) ∈ {(x+1, y+1, z), (x-1, y+1, z)}, ou seja, se δ(SHALLOW_BESIDE) = {(1,1,0), (-1,1,0)}; como ilustrado na Figura 6-15 abaixo:

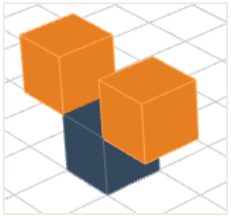

Figura 6-15 SHALLOW_BESIDE

6.3.2 DEEP_BESIDE

Uma posição *p1* será considerada ao lado profundo de (do inglês, *deep beside*) outra posição *p2* se e somente se location(*p1*) = (x, y, z) e location(*p2*) ∈ {(x+1, y+3, z), (x-1, y+3, z)}, ou seja, se δ(DEEP_BESIDE) = {(1,3,0), (-1,3,0)}; como ilustrado na Figura 6-16 abaixo:

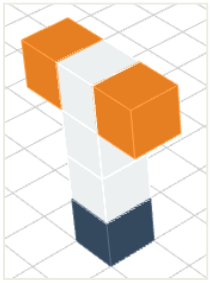

Figura 6-16 DEEP_BESIDE

6.3.3 SHALLOW_FACING

Uma posição *p1* será considerada de frente raso de (do inglês, *shallow facing*) outra posição *p2* se e somente se location(*p1*) = (x, y, z) e location(*p2*) ∈ {(x, y+1, z+1), (x, y+1, z-1)}, ou seja, se δ(SHALLOW_FACING) = {(0,1,1), (0,1,-1)}; como ilustrado na Figura 6-17 abaixo:

Figura 6-17 SHALLOW_FACING

6.3.4 DEEP_FACING

Uma posição *p1* será considerada de frente profundo de (do inglês, *deep facing*) outra posição *p2* se e somente se location(*p1*) = (x, y, z) e location(*p2*) ∈ {(x, y+3, z+1), (x, y+3, z-1)}, ou seja, se δ(DEEP_FACING) = {(0,3,1), (0,3,-1)}; como ilustrado na Figura 6-18 abaixo:

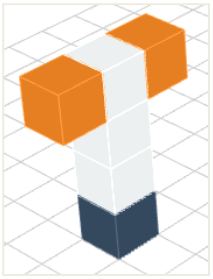

Figura 6-18 DEEP_FACING

6.3.5 ABOVE3

Uma posição *p1* será considerada três-acima de (do inglês, *three above*) outra posição *p2* se e somente se location(*p1*) = (x, y, z) e location(*p2*) = (x, y-3, z), ou seja, se δ(ABOVE3) = {(0,-3,0)}; como ilustrado na Figura 6-19 abaixo:

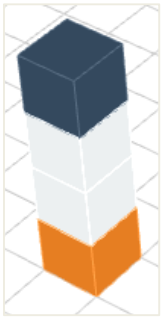

Figura 6-19 ABOVE3

6.3.6 ABOVE4

Uma posição *p1* será considerada quatro-acima de (do inglês, *four above*) outra posição *p2* se e somente se location(*p1*) = (x, y, z) e location(*p2*) = (x, y-4, z), ou seja, se δ(ABOVE4) = {(0,-4,0)}; como ilustrado na Figura 6-20 abaixo:

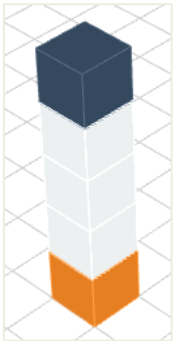

Figura 6-20 ABOVE4

6.3.7 BEFORE

Uma posição *p1* será considerada antes de (do inglês, *before*) outra posição *p2* se e somente se location(*p1*) = (x, y, z) e location(*p2*) = (x+1, y, z), ou seja, se δ(BEFORE) = {(1,0,0)}; como ilustrado na Figura 6-21 abaixo:

Figura 6-21 BEFORE

6.3.8 UNDER4

Uma posição *p1* será considerada quatro-abaixo de (do inglês, *four under*) outra posição *p2* se e somente se location(*p1*) = (x, y, z) e location(*p2*) = (x, y+4, z), ou seja, se δ(UNDER4) = {(0,4,0)}; como ilustrado na Figura 6-22 abaixo:

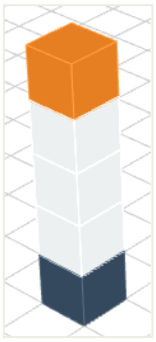

Figura 6-22 UNDER4

6.3.9 Gramática

A gramática do poço d'água:

S → **SP** WELL

WELL → BORDER **DEEP_BESIDE** BODY

BORDER → X_EDGE **BESIDE** Z_EDGE **FACING** X_EDGE **BESIDE** Z_EDGE

X_EDGE → *gravel* **BESIDE** *gravel* **BESIDE** *gravel* **BESIDE** *gravel* **BESIDE** *gravel*

Z_EDGE → *gravel* **FACING** *gravel* **FACING** *gravel* **FACING** *gravel* **FACING** *gravel*

BODY → X_SIDE **DEEP_BESIDE** Z_SIDE **DEEP_FACING** X_SIDE **DEEP_BESIDE** Z_SIDE **BEFORE** CORE **UNDER4** ROOF

X_SIDE → CORNER **SHALLOW_BESIDE** ACCESS **SHALLOW_BESIDE** ACCESS

Z_SIDE → CORNER **SHALLOW_FACING** ACCESS **SHALLOW_FACING** ACCESS

CORNER → *fence* **ABOVE** *fence* **ABOVE** *cobblestone* **ABOVE** *cobblestone*

ACCESS → *cobblestone* **ABOVE** *cobblestone*

CORE → *water* **BESIDE** *water* **FACING** *water* **BESIDE** *water*

ROOF → *cobblestone* **AROUND** *cobblestone* **AROUND** *cobblestone* **AROUND** *cobblestone* **AROUND** *cobblestone* **AROUND** *cobblestone* **AROUND** *cobblestone* **AROUND** *cobblestone* **AROUND** *cobblestone* **AROUND** *cobblestone* **AROUND** *cobblestone* **AROUND** *cobblestone* **AROUND** *cobblestone* **AROUND** *cobblestone* **AROUND** *cobblestone*

SP → ε

BESIDE → ε

FACING → ε

DEEP_BESIDE → ε

DEEP_FACING → ε

LOW_BESIDE → ε

LOW_FACING → ε

ABOVE → ε

ABOVE4 → ε

ABOVE3 → ε

AROUND → ε

UNDER → ε

NEAR → ε

O caminho de leitura do poço d'água tem seu início pelo anel de blocos *gravel* e em seguida passa para o interior da estrutura. É feita, então, a leitura de cada lado com uma varredura vertical das colunas. Em seguida é lido o núcleo de *water*. E, por fim, é lido o teto.

Outra diferença entre o poço d'água e as estruturas apresentadas antes dele se dá pelo fato de a gramática do poço aceitar estruturas com certas partes deslocadas, seu teto, por exemplo. Isso ocorre pela forma abrangente com que a relação posicional NEAR foi

definida. Para garantir a integridade da entrada e o correto alinhamento da estrutura sem alterar a gramática e as relações posicionais, seria preciso fazer verificações com as ações do analisador nos momentos das reduções. O mesmo acontece em linguagens de programação na verificação de que um identificador, como o nome de uma variável ou função, por exemplo, já foi declarado.

6.4 Casa Tipo 1

Existem cerca de dez tipos de estruturas que podem ser chamadas de casas geradas automaticamente pelo Minecraft. Foram selecionadas duas das mais simples para demostrar a capacidade e o funcionamento das gramaticas posicionais. O primeiro tipo de casa apresentado é constituído de colunas e chão de pedra. Possui paredes e teto de madeira, e seu espaço interno é um cubo de lado três. Mais formalmente, foi considerado casa tipo 1 (Figura 6-23) o conjunto de blocos do chão formado por nove blocos tipo *cobblestone* dispostos horizontalmente num quadrado de dimensões três por três, dos pilares formados por quatro blocos *cobblestone* empilhados verticalmente e posicionados diagonalmente alinhados em cada vértice do chão; pelas colunas de paredes formadas por três blocos *planks* (Figura 6-24) empilhados sobre um bloco *cobblestone* e sob um bloco *log* dispostas entre dois pilares e contendo uma porta *wooden_door* (Figura 6-25) ou janela *glass_pane* (Figura 6-26) em seu centro; e finalmente pelo teto formado por blocos tipo *planks*. Observe que *wooden_door* é na verdade um par de blocos empilhados verticalmente, e ambos os blocos recebem a classificação de *wooden_door*.

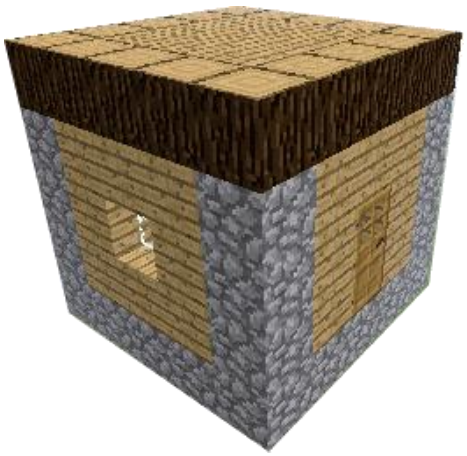

Figura 6-23 Casa Tipo 1

Figura 6-24
Bloco planks

Figura 6-25 Blocos
wooden_door

Figura 6-26
Bloco *glass_pane*

6.4.1 DEEPER_BESIDE

Uma posição *p1* será considerada ao lado mais profundo de (do inglês, *deeper beside*) outra posição *p2* se e somente se location(*p1*) = (x, y, z) e location(*p2*) ∈ {(x+1, y+4, z), (x-1, y+4, z)}, ou seja, se δ(DEEPER_BESIDE) = {(1,4,0), (-1,4,0)}; como ilustrado na Figura 6-27 abaixo:

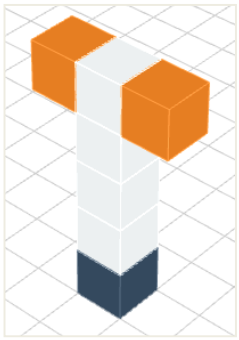

Figura 6-27 DEEPER_BESIDE

6.4.2 DEEPER_FACING

Uma posição *p1* será considerada de frente mais profundo de (do inglês, *deeper facing*) outra posição *p2* se e somente se location(*p1*) = (x, y, z) e location(*p2*) ∈ {(x, y+4, z+1), (x, y+4, z-1)}, ou seja, se δ(DEEPER_FACING) = {(0,4,1), (0,4,-1)}; como ilustrado na Figura 6-28 abaixo:

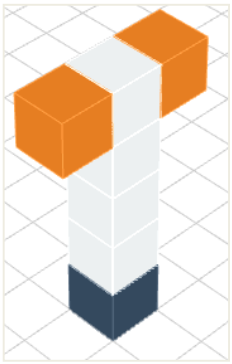

Figura 6-28 DEEPER_FACING

6.4.3 Gramática

A gramática da casa tipo 1:

S → **SP** HOUSE

HOUSE → WALLS **AROUND** FLOOR **UNDER4** ROOF

WALLS → X_SIDE **DEEPER_BESIDE** Z_SIDE **DEEPER_FACING** X_SIDE **DEEPER_BESIDE** Z_SIDE

X_SIDE → PILLAR **DEEPER_BESIDE** WALL **DEEPER_BESIDE** ACCESS **DEEPER_BESIDE** WALL

Z_SIDE → PILLAR **DEEPER_FACING** WALL **DEEPER_FACING** ACCESS **DEEPER_FACING** WALL

PILLAR → *log* **ABOVE** *cobblestone* **ABOVE** *cobblestone* **ABOVE** *cobblestone* **ABOVE** *cobblestone*

WALL → *log* **ABOVE** *planks* **ABOVE** *planks* **ABOVE** *planks* **ABOVE** *cobblestone*

ACCESS → DOOR

ACCESS → WINDOW

DOOR → *log* **ABOVE** *planks* **ABOVE** *wooden_door* **ABOVE** *wooden_door* **ABOVE** *cobblestone*

WINDOW → *log* **ABOVE** *planks* **ABOVE** *glass_pane* **ABOVE** *planks* **ABOVE** *cobblestone*

FLOOR → FLOOR_LINE **AROUND** FLOOR_LINE **AROUND** FLOOR_LINE

FLOOR_LINE → *cobblestone* **AROUND** *cobblestone* **AROUND** *cobblestone*

ROOF → ROOF_LINE **AROUND** ROOF_LINE **AROUND** ROOF_LINE

ROOF_LINE → *planks* **AROUND** *planks* **AROUND** *planks*

SP → ε

ABOVE → ε

BESIDE → ε

FACING → ε

NEAR → ε

UNDER → ε

A leitura dessa estrutura é semelhante à da parte interior do poço d'água, onde é feita uma varredura vertical das colunas das paredes e por fim o chão e teto são lidos horizontalmente. Também semelhante ao poço d'água é a necessidade do uso de ações para garantir o alinhamento das partes entre si, ou seja, do teto e chão com as paredes. Há ainda outra verificação que pode ser feita, esta relativa à presença de portas e janelas, que possuem posição definida em cada parede, mas a gramática não restringe a existência das diversas combinações possíveis entre esses acessos.

6.5 Casa Tipo 2

O segundo tipo de casa difere do primeiro por seu tamanho reduzido em um bloco de largura, suas colunas de blocos tipo *log* e seu chão de blocos tipo *dirt*. A casa tipo 2 (Figura 6-29) não possui área interna quadrada e tem suas paredes de frente e fundos sobre os lados curtos. As paredes sobre os lados longos são idênticas às da casa tipo 1. Suas colunas têm altura um bloco menor que a das paredes.

Figura 6-29 Casa Tipo 2

6.5.1 LOW_BESIDE

Uma posição *p1* será considerada ao lado baixo de (do inglês, *low beside*) outra posição *p2* se e somente se location(*p1*) = (x, y, z) e location(*p2*) ∈ {(x+1, y-3, z), (x-1,

y-3, z)}, ou seja, se δ(LOW_BESIDE) = {(1,-3,0), (-1,-3,0)}; como ilustrado na Figura 6-30 abaixo:

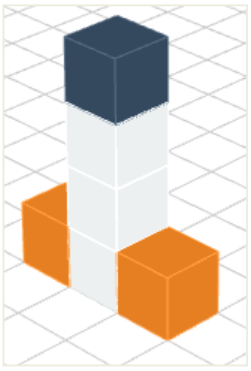

Figura 6-30 LOW_BESIDE

6.5.2 HIGH_BESIDE

Uma posição *p1* será considerada ao lado alto de (do inglês, *high beside*) outra posição *p2* se e somente se location(*p1*) = (x, y, z) e location(*p2*) ∈ {(x+1, y-4, z), (x-1, y-4, z)}, ou seja, se δ(HIGH_BESIDE) = {(1,-4,0), (-1,-4,0)}; como ilustrado na Figura 6-31 abaixo:

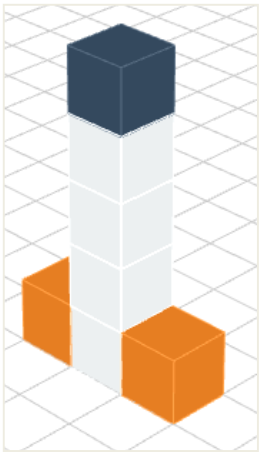

Figura 6-31 HIGH_BESIDE

6.5.3 LOW_FACING

Uma posição *p1* será considerada baixo de frente de (do inglês, *low facing*) outra posição *p2* se e somente se location(*p1*) = (x, y, z) e location(*p2*) ∈ {(x, y-3, z+1), (x, y-3, z-1)}, ou seja, se δ(LOW_FACING) = {(0,-3,1), (0,-3,-1)}; como ilustrado na Figura 6-32 abaixo:

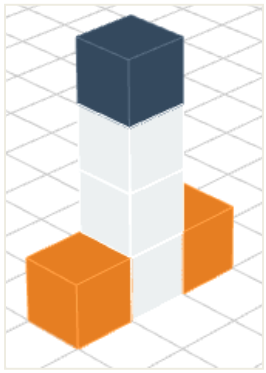

Figura 6-32 LOW_FACING

6.5.4 HIGH_FACING

Uma posição *p1* será considerada alto de frente de (do inglês, *high facing*) outra posição *p2* se e somente se location(*p1*) = (x, y, z) e location(*p2*) ∈ {(x, y-4, z+1), (x, y-4, z-1)}, ou seja, se δ(HIGH_FACING) = {(0,-4,1), (0,-4,-1)}; como ilustrado na Figura 6-33 abaixo:

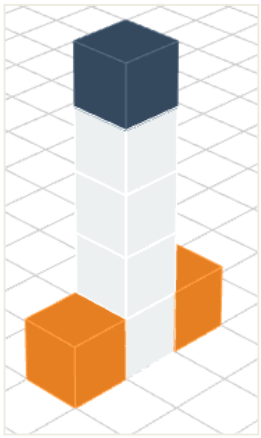

Figura 6-33 HIGH_FACING

6.5.5 Gramática

A gramática da casa tipo 2:

S → **SP** HOUSE

HOUSE → WALLS **AROUND** ROOF **ABOVE4** FLOOR

WALLS → X_SIDE **LOW_BESIDE** Z_SIDE **LOW_FACING** X_SIDE
LOW_BESIDE Z_SIDE

X_SIDE → PILLAR **HIGH_BESIDE** WALL **HIGH_BESIDE** ACCESS
HIGH_BESIDE WALL

X_SIDE → PILLAR **HIGH_BESIDE** WALL **HIGH_BESIDE** WALL

X_SIDE → PILLAR **HIGH_BESIDE** ACCESS **HIGH_BESIDE** WALL

Z_SIDE → PILLAR **HIGH_FACING** WALL **HIGH_FACING** ACCESS **HIGH_FACING** WALL

Z_SIDE → PILLAR **HIGH_FACING** WALL **HIGH_FACING** WALL

Z_SIDE → PILLAR **HIGH_FACING** ACCESS **HIGH_FACING** WALL

PILLAR → *cobblestone* **UNDER** *log* **UNDER** *log* **UNDER** *log*

WALL → *cobblestone* **UNDER** *planks* **UNDER** *planks* **UNDER** *planks* **UNDER** *log*

ACCESS → DOOR

ACCESS → WINDOW

DOOR → *cobblestone* **UNDER** *wooden_door* **UNDER** *wooden_door* **UNDER** *planks* **UNDER** *log*

WINDOW → *cobblestone* **UNDER** *planks* **UNDER** *glass_pane* **UNDER** *planks* **UNDER** *log*

ROOF → *log* **AROUND** ROOF

ROOF → *log*

FLOOR → *dirt* **AROUND** FLOOR

FLOOR → *dirt*

SP → ε

UNDER → ε

BESIDE → ε

FACING → ε

LOW_BESIDE → ε

LOW_FACING → ε

NEAR → ε

ABOVE → ε

É preciso observar que a leitura das paredes ocorre verticalmente, porém em sentido oposto aos da casa tipo 1. E, da mesma forma que a plantação, essa gramática só reconhece casas tipo 2 orientadas no eixo z. Para efetuar a leitura de casas tipo 2 orientadas no eixo x é preciso adaptar a gramática como indicado no caso da plantação. A leitura de casas ou estruturas análogas não limitadas aos tamanhos aqui descritos pode ser feita com o uso de ações para validar a integridade da construção verificando o encaixe entre suas partes (como paredes, teto e chão).

6.6 Segmento

Em geral, as estruturas são compostas por blocos adjacentes de uma lista pequena de tipos. Então um forma de sondar o espaço em busca de possíveis formações estruturais é usar uma pequena gramática de segmento. Os tipos de blocos mais característicos nas formas estruturais são os seguintes: *cobblestone*, *log*, *planks*, *gravel* e *fence*. Um segmento é formado por quaisquer dois blocos dentre estes tipos dispostos a distância de até uma posição em cada dimensão. Um analisador para esta gramática serviria para sondar, durante uma varredura do espaço, por exemplo, uma posição como sendo o início de outra estrutura mais complexa descrita por outra gramática.

6.6.1 SURROUNDING

Uma posição *p1* será considerada em torno de (do inglês, *surrounding*) outra posição *p2* se e somente se location(*p1*) = (x, y, z) e location(*p2*) = (k, u, s), onde k ∈ [x-1, x+1], u ∈ [y-1, y+1] e s ∈ [z-1, z+1], como ilustrado na Figura 6-34 abaixo, onde o bloco azul-escuro da posição inicial encontra-se no centro deste cubo 3x3 e posições candidatas em laranja-claro:

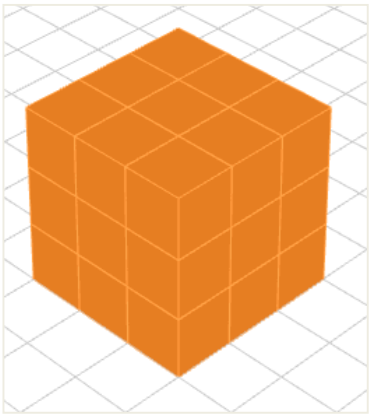

Figura 6-34 SURROUNDING

6.6.2 Gramática

S → **SP** PAIR

PAIR → BLOCK **SURROUNDING** BLOCK

BLOCK → *cobblestone*

BLOCK → *log*

BLOCK → *planks*
BLOCK → *gravel*
BLOCK → *fence*
SP → ε
SURROUNDING → ε

Desse modo é possível analisar uma posição de uma forma rápida pra decidir se essa posição pode, potencialmente, participar de uma estrutura. Em caso positivo, uma análise mais detalhada deve ser feita com a finalidade de identificar que estrutura pode ter essa posição como integrante.

6.7 Estruturas de tamanho variável em mais dimensões

Algumas das estruturas apresentadas neste capítulo devem possuir as medidas exatas em número de blocos em cada trecho para que sejam aceitas pelo analisador da gramática posicional correspondente. Esse é o caso do poço d'água e de ambas as casas (seções 6.3, 6.4 e 6.5). Os exemplos da árvore (6.1) e da plantação (6.2), por outro lado, permitem que suas estruturas tenham tamanhos variáveis em uma direção. Uma árvore indefinidamente alta, ou com uma copa indefinidamente volumosa seria aceita pelo analisador da gramática em 6.1.2. E, da mesma forma, uma plantação de largura fixa na dimensão *z* porém de comprimento indefinido na dimensão *x* seria analogamente aceita pelo analisador da gramática em 6.2.4.

Como indicado anteriormente, o analisador léxico espacial efetua um caminho linear de leitura sobre o espaço de *voxels*. O caminho é dito linear pois é formado por uma sequência dos blocos lidos um a um. Esse formato de leitura deve ser mantido na análise de estruturas de tamanho variável em mais de uma dimensão, como em uma plantação que cresça tanto na dimensão *x* quanto na *z*. Para que uma gramática posicional livre de contexto descreva uma linguagem de estruturas de tamanho variável em mais de uma dimensão, é necessário que ela possua mais de uma regra de produção recursiva, como no seguinte formato:

A → A **REL** *b*
A → *c*

Onde A é um símbolo não terminal, REL é uma relação posicional, *b* e *c* são símbolos terminais, e *c* pode ser igual a ε ou a qualquer outro símbolo terminal. Além disso, é importante que as relações posicionais de diferentes regras de produção recursivas atuem sobre dimensões diferentes. Atendidas essas condições, um analisador gramatical nesses termos não encontrará dificuldade em analisar estruturas de tamanho variável em mais de uma dimensão. O problema surge quando é necessário que esses tamanhos obedeçam certas restrições de proporção, consistência ou integridade. Considere, por exemplo, uma cerca retangular para uma fazenda ou um portal de acesso a uma construção; essas estruturas só serão válidas se forem contínuas e formarem um perímetro em volta de uma área vazia, mesmo que em algumas faces elas sejam ladeadas por outra formação, como uma parede ou chão.

No âmbito das gramáticas livres de contexto, as derivações de uma regra de produção são isoladas das demais derivações das outras regras. Por conta disso, não é possível apenas com gramáticas livres de contexto garantir que partes do produto das derivações estejam relacionadas entre si, afinal, isso é condição necessária para as gramáticas livres de contexto. Para que seja possível analisar e aceitar uma estrutura de tamanho variável em mais de uma dimensão e que possua uma restrição de consistência ou proporção, é preciso fazer uso de gramáticas com ações.

Considere novamente o caso da plantação, em que a largura na dimensão *z* é fixa e a quantidade de 'linhas de plantação' no comprimento é variável sobre dimensão *x*. Uma das formas de tornar esse caso numa gramática que descreva tamanhos variáveis em ambas as dimensões (*x* e *z*) é:

1. Tornar recursiva a regra de 'linha de plantação' (ROW em 6.2.4);
2. Abandonar a relação posicional FOLLOWED_BY (em 6.2.3) e substituí-la por outra capaz de fornecer a próxima posição de leitura com distância variável a partir da última, a VAR_FOLLOWED_BY abaixo; e
3. Incluir ações às regras de produção da gramática para que seja efetuada a verificação de integridade.

A relação posicional FOLLOWED_BY (em 6.2.3) aponta como posição candidata à próxima posição de leitura aquela que está a um passo positivo na dimensão *x* e a seis negativos na *z*, ou seja, δ(FOLLOWED_BY) = {(1,0,-6)}. Existe mais de uma forma de tornar variável essa quantidade em *z*. Uma delas é fixar em uma variável o componente *z*

da posição inicial e, assim, a relação posicional VAR_FOLLOWED_BY faria uso dessa variável para garantir que a próxima leitura da 'linha de plantação' coincida com o alinhamento correto na dimensão *z*. As ações adicionadas às regras de produção e os operadores posicionais neste exemplo terão permissão de leitura e escrita a esta variável. Ela será denotada por S_z e armazenará o valor da componente *z* da posição inicial de leitura.

6.7.1 VAR_FOLLOWED_BY

Uma posição *p1* será considerada variavelmente seguida de (do inglês, *variably followed by*) outra posição *p2* se e somente se location(*p1*) = (x, y, z) e location(*p2*) = (x+1, y, S_z). As ilustrações dessa relação posicional seriam semelhantes à da Figura 6-10 FOLLOWED_BY variando a componente *z* do bloco claro (laranja).

6.7.2 Gramática

Assim como a gramática da plantação (em 6.2.4), esta descreverá uma sequência de linhas de plantação dispostas ao longo da dimensão *z* e lado a lado com demais linhas na direção da dimensão *x*. A diferença entre essas gramáticas reside na possível variação de comprimento das linhas de plantação descrita abaixo:

S → **SP** PLANTATION
PLANTATION → ENDING **VAR_FOLLOWED_BY** FILLING
VAR_FOLLOWED_BY ENDING
ENDING → ENDING **BEHIND** *log*
ENDING → *log*
FILLING → FILLING **VAR_FOLLOWED_BY** ROW
FILLING → ROW
ROW → *log* **BEHIND** CORE **ASIDE** *log*
CORE → CORE **ASIDE** STRETCH
CORE → STRETCH
STRETCH → PLANTS **ASIDE** IRRIGATION **BEHIND** PLANTS
PLANTS → PLANT **ASIDE** PLANT
PLANT → *farmland* **UNDER** *wheat*
IRRIGATION → *water*

SP → ε { escreva o valor da componente *z* da posição inicial em S_z }

VAR_FOLLOWED_BY → ε

ASIDE → ε

UNDER → ε

BEHIND → ε

Dessa forma, ao ler o primeiro bloco tipo *log*, o analisador léxico causará no analisador sintático uma redução da regra SP → ε e consequentemente a execução de seu bloco de ações associadas. Assim, a componente *z* da posição inicial estará preservada em S_z para que, quando o operador posicional VAR_FOLLOWED_BY for acionado este possa indicar a posição do primeiro bloco da próxima linha de plantação com componentes *x* e *z* corretos. Assim uma plantação de comprimento e largura variáveis pode ser analisada e aceita por analisadores de gramáticas posicionais livres de contexto.

Capítulo 7 - Implementação

Este capítulo trata dos detalhes tecnológicos da implementação da solução proposta. Nele são descritas as bibliotecas, pacotes e plataformas utilizadas, assim como as interfaces para interação com o analisador léxico espacial e o bot.

Visando facilitar a implementação e verificação do funcionamento das gramáticas posicionais, foi escolhida a plataforma NodeJS (TILKOV; VINOSKI, 2010) da linguagem JavaScript (GOODMAN, 2007) por apresentar a melhor seleção de bibliotecas capazes de realizar as tarefas necessárias. Para geração dos analisadores gramaticais, foi utilizada a biblioteca Jison (CARTER, 2014), que é uma versão portada do famoso Bison (AABY, 2003). Para conexão e interface com o Minecraft, foi utilizada a biblioteca MineFlayer (KELLEY et al., 2015). Ela viabiliza a criação de *bots* e é capaz de disponibilizar total acesso às informações necessárias e ações desejáveis dentro do jogo.

7.1 O analisador léxico espacial

A implementação do analisador léxico espacial foi feita em spatial-jison-lex (SANTOS, 2015). Essa implementação é compatível com a biblioteca Jison (CARTER, 2014). A instanciação de um analisador léxico nesta implementação requer 3 (três) parâmetros: o analisador sintático, as relações posicionais e uma ordem de precedência das relações posicionais. O primeiro parâmetro – o analisador sintático – deve ser uma instância fornecida pelo Jison, ou equivalente. Dela serão lidas as regras de produção da gramática e os símbolos do lado direito de cada regra, após um pré-processamento realizado pela construção do analisador sintático. O segundo parâmetro contém uma lista das relações posicionais e um dado que caracterize seu significado, podendo ser tanto um operador posicional em forma de função ou uma lista de vetores de diferenças posicionais, como definido por δ. No processo de descoberta da próxima posição de leitura, essas informações das relações posicionais serão utilizadas para cálculo das posições candidatas. O terceiro parâmetro é opcional e, se fornecido, determina uma ordem

específica de prioridade em que as relações posicionais devem ser acionadas para geração das posições candidatas.

A interação entre os analisadores (léxico e sintático) no processo de leitura se dá da seguinte forma:

1. Analisador sintático é acionado para a análise e recebe a entrada;
2. Analisador sintático aciona o analisador léxico, passando-lhe a entrada recebida;
3. Analisador léxico recebe a entrada e configura-se para iniciar uma nova leitura sobre esta;
4. Analisador sintático solicita um novo símbolo ao analisador léxico;
5. Analisador léxico procede ao descrito no Algoritmo 5-1 para obter a nova posição de leitura, ler o símbolo e fornecê-lo ao analisador sintático, caso o símbolo seja encontrado;
6. Analisador sintático recebe o símbolo fornecido pelo analisador léxico e procede sua análise;
7. As etapas de número 4 a 6 desta lista são repetidas até que o analisador léxico não seja mais capaz de ler símbolos da entrada;
8. Indicado o fim da leitura, o analisador léxico deve decidir por aceitar a entrada ou acusar um erro.

No caso do analisador léxico tradicional, a leitura da entrada é feita em uma direção e sentido apenas, geralmente da esquerda para a direita. O analisador léxico espacial, entretanto, faz uso das relações posicionais para proceder a leitura sobre a entrada. Além disso, o spatial-jison-lex requer um gerente de leitura da entrada que atenda uma interface projetada para suprir as informações necessárias para a análise léxica.

7.2 A Interface de Leitura da Entrada

A interface de leitura da entrada objetiva normalizar a troca de informações entre o spatial-jison-lex e os diversos espaços de entrada possíveis. Assim a implementação do analisador léxico espacial não depende de nenhuma forma das bibliotecas e protocolos relacionados ao Minecraft.

Estão abaixo listadas as funções que devem ser implementadas por um módulo de gerência de entrada para que este seja compatível com o spatial-jison-lex:

- **Has:** função que recebe uma posição como parâmetro e retorna um valor booleano indicando se a posição existe como um valor válido a ser endereçado para leitura;
- **Get:** função que recebe uma posição como parâmetro e retorna seu conteúdo;
- **Visit:** função que recebe uma posição como parâmetro e procede uma visitação sobre a posição; nenhum retorno é esperado;
- **Key:** função que recebe uma posição como parâmetro e retorna um identificador único e serializável, tipicamente, uma *string*;
- **Token:** função que recebe uma posição como parâmetro e retorna um símbolo a ser considerado na passagem para o analisador sintático.

Uma implementação dessa interface foi desenvolvida para viabilizar o acesso do spatial-jison-lex aos dados dos blocos do Minecraft. Para que o spatial-jison-lex seja utilizado em um ambiente diferente do abordado neste trabalho, basta que uma nova implementação gerente de entrada seja concebida respeitando a interface acima.

7.3 O Bot

Além do analisador léxico e das gramáticas posicionais das estruturas escolhidas, foi também desenvolvido um simples *bot* para interligar os demais componentes. Este *bot* dispõe dos analisadores das gramáticas posicionais e de suas relações posicionais correspondentes, além de estar ligado ao gerente de leitura da entrada. Ele tem a capacidade de investigar cada posição no espaço tridimensional em que está inserido. Quando acionada sua rotina de reconhecimento das redondezas, ele inicia uma varredura pelo espaço e testa cada voxel como posição inicial de cada gramática que possui. Durante a execução, cada analisador visitará as posições relevantes à sua análise, segundo suas relações posicionais e decidirá se há uma estrutura conhecida a partir da posição inicial fornecida. Se for este o caso, o analisador sintático sinaliza ao *bot*, e este anuncia o objeto reconhecido.

Sobre a plataforma fornecida pelo MineFlayer, representada pelo módulo *Client API*, foi desenvolvido o módulo de tratamento de mensagens entre jogadores (*Chat*

Message Handler). Ele tem o papel de acionar o módulo de análise espacial (*Spatial Parser*) de acordo com os comandos recebidos pelas mensagens. Para efetuar a análise, o *Spatial Parser* utiliza os analisadores das gramáticas disponíveis no módulo *Positional Grammars* e acessa os blocos do espaço através do módulo *Client API.* Terminada a análise, uma mensagem é enviada pelo *Chat Message Handler* através do *Client API* para o servidor do jogo. O bot tem a capacidade de responder a comandos básicos destinados à demonstração do funcionamento das gramáticas posicionais concebidas e sua consequente habilidade de reconhecer objetos tridimensionais no ambiente do jogo. São listados abaixo os comandos suportados por esta implementação de *bot*:

- **Análise Singular:** Identificação de objetos a partir de uma posição inicial indicada. Cada analisador gramatical disponível é usado para avaliar a posição fornecida, e, caso o resultado da análise seja positivo, o *bot* anuncia publicamente o tipo de estrutura identificado.
- **Varredura:** Execução de uma varredura pelo espaço, iniciando-a pelo chão, seguindo verticalmente para cima aumentando gradativamente o raio da busca. Para cada posição investigada o bot utiliza o analisador da gramática de segmento (6.6.2). Caso a análise seja positiva, a posição em questão será avaliada segundo cada analisador construído a partir de cada gramática posicional. Caso uma dada posição não corresponda ao objeto do analisador, a análise falhará imediatamente. Caso corresponda, a análise prossegue para os demais blocos até que o analisador aceite ou rejeite o objeto. Todo objeto reconhecido com sucesso é então anunciado publicamente pelo *bot*.

A Figura 7-1 abaixo ilustra o diagrama de módulos componentes do *bot*. A Figura 7-2 ilustra um exemplo de fluxo de comunicação entre os módulos. Neste exemplo outro jogador envia uma mensagem *chat* por seu cliente Minecraft. A mensagem chega ao servidor Minecraft e é encaminhada aos demais jogadores conectados. O módulo *Chat (Message) Handler* do *bot* recebe e trata a mensagem, produzindo uma chamada ao módulo *Spatial Parser*. Este, por sua vez, faz uso dos analisadores das gramáticas posicionais que possui e requisita ao servidor as informações dos blocos pertinentes. Isto é feito por meio do módulo *Client API*, abstraído nesta ilustração por simplicidade. A análise é feita sobre as informações dos blocos, e o resultado é transmitido para o *Chat*

Handler e posteriormente para o servidor, que encaminha a mensagem para os demais jogadores conectados.

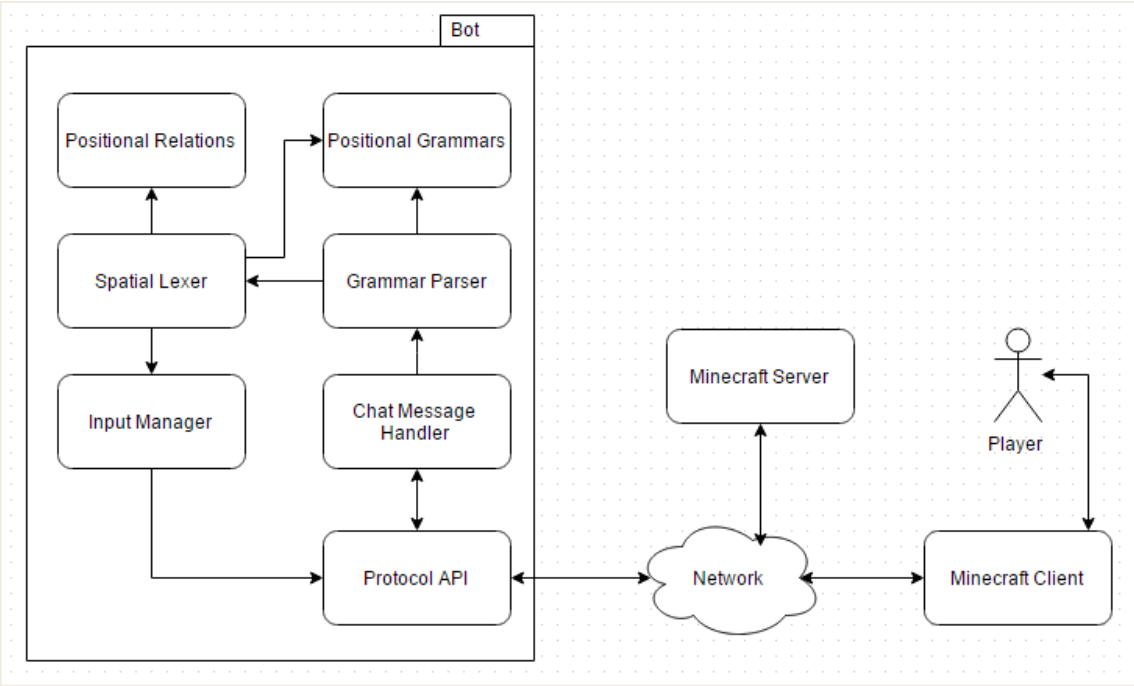

Figura 7-1 Diagrama de módulos do bot

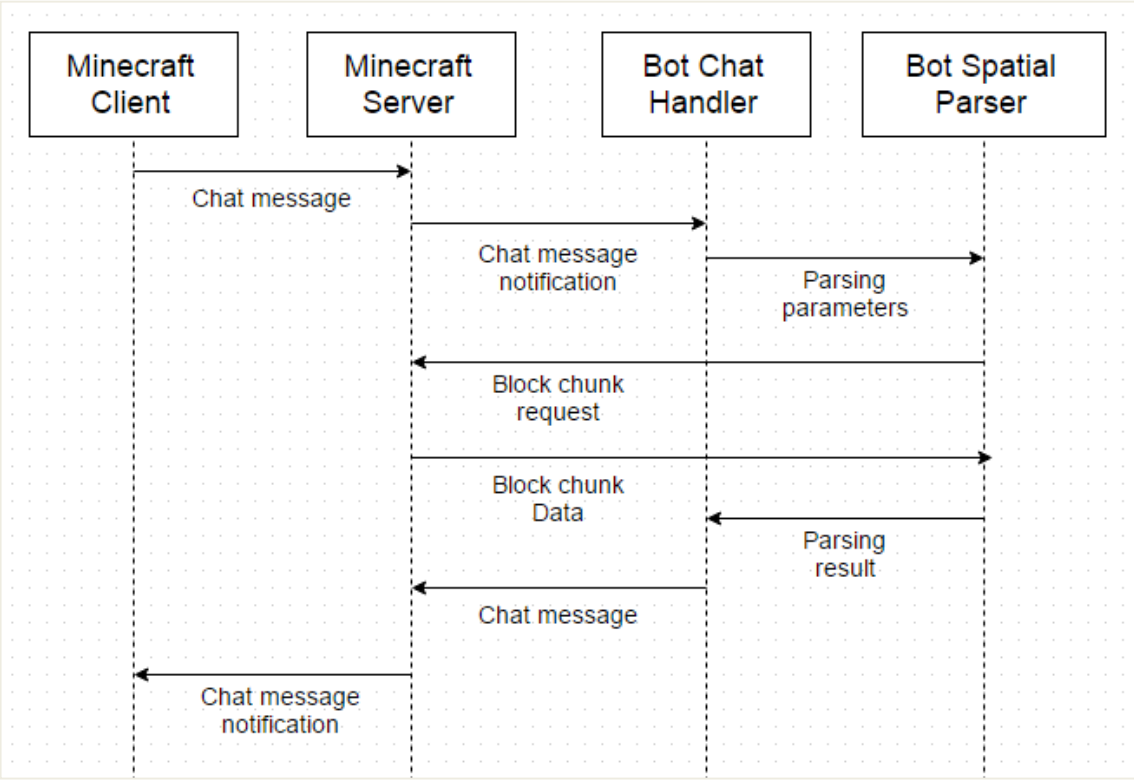

Figura 7-2 Diagrama de fluxo de mensagens com o bot

Este simples *bot* não foi equipado com inteligência artificial. Seu objetivo é ilustrar o funcionamento das análises léxica e sintática sobre estruturas de voxels no espaço tridimensional. Demonstrada a capacidade de tratar os blocos do espaço e reconhecer neles estruturas de interesse, é possível então acoplar um módulo responsável por inteligência. Este terá a sua disposição como informação de entrada não apenas as posições e tipos dos blocos à sua volta, mas também a informação extra de que certos conjuntos de blocos estão organizados segundo as regras de uma gramática posicional livre de contexto.

Capítulo 8 - Análise de Desempenho

Este capítulo trata dos experimentos conduzidos em ambiente de teste com a finalidade de medição e análise de desempenho da implementação da solução proposta. Além disso, os resultados são apresentados e discutidos.

A fim de verificar a eficiência performática da solução desenvolvida, os seguintes experimentos foram concebidos:

I. Medição do tempo de análise de uma estrutura conhecida;
II. Medição do tempo de análise e classificação de uma estrutura desconhecida; e
III. Medição do tempo de reconhecimento de uma região preenchida com estruturas.

Cada experimento foi executado no total de 1000 (mil) vezes. Os tempos foram medidos em milissegundos, e em cada tabela de resultados são expostas as média dos tempos, o desvio padrão e a razão do segundo sobre o primeiro como medida de precisão. Todos os experimentos foram realizados no espaço ilustrado pela Figura 8-1.

Figura 8-1 Região usada nos experimentos

8.1 Ambiente

Os experimentos foram executados em um computador portátil com a descrição abaixo:

Tabela 8-1 Características do ambiente

Característica	Valor
Processador	Intel® Core™ i7-3632QM 2.20GHz
Memória RAM	8,00 GB DDR3-1600
Disco Rígido	1TB 5400 RPM 8MB Cache SATA, 32GB SSDR (RAID 1)
Sistema Operacional	Microsoft® Windows™ 8 64bits
NodeJS	Versão 0.12.0
Minecraft	Versão 1.8.3
MineFlayer	Versão 1.2.1
Jison	Versão 0.4.15

8.2 Resultados

8.2.1 Experimento I

Neste experimento, uma mesma estrutura é analisada várias vezes pelo *bot*, sendo que tanto a posição inicial quanto a classe da estrutura são dados. De posse da classe da estrutura, o bot pode simplesmente utilizar o analisador correspondente à gramática dessa classe. Não é feita nenhuma busca por qual gramática usar. Dessa forma, espera-se medir o tempo de sondagem das posições candidatas, realizada pelo analisador léxico espacial, junto com o tempo da análise sintática. A complexidade desse procedimento é também $\Theta(n\ r\ p\ \Theta(a))$; onde n é o número de posições candidatas dadas por uma relação posicional; r é o número de relações posicionais; p é o número de posições em um objeto de entrada; e $\Theta(a)$ é a complexidade de acessar uma posição na entrada.

Tabela 8-2 Resultados do Experimento I

Estrutura	Média (ms)	Desvio Padrão (ms)
Árvore	3,522	0,628
Plantação	5,716	1,032
Poço d'água	16,039	1,457
Casa Tipo 1	11,801	1,414
Casa Tipo 2	10,855	1,322

Uma análise dos dados permite concluir que o tempo de processamento de uma dada estrutura é diretamente proporcional ao tamanho em blocos da estrutura. Estruturas maiores levam mais tempo para serem lidas e analisadas.

8.2.2 Experimento II

No segundo experimento, a informação sobre a classe é removida, porém a posição inicial de leitura é mantida. Assim, o *bot* deve utilizar cada analisador de gramática posicional para efetuar a análise da posição inicial dada e assim determinar a classe da estrutura fornecida. Foi medido o tempo desde o início da análise por uma gramática até a identificação da estrutura pelo analisador correto. Dessa forma, espera-se medir também o tempo gasto nas tentativas feitas por cada analisador gramatical. Isto, claro, somado aos tempos de sondagem de posições e análise sintática. A complexidade desse procedimento é semelhante à do experimento anterior, sendo incluída apenas a quantidade t de tipos de estruturas conhecidas, ficando assim com a complexidade $\Theta(t\ n\ r\ p\ \Theta(a))$; onde os demais termos têm a mesma representação do que no Experimento I.

Tabela 8-3 Resultados do Experimento II

Estrutura	Média (ms)	Desvio Padrão (ms)
Árvore	3,813	0,805
Plantação	7,055	1,186
Poço d'água	16,376	1,427
Casa Tipo 1	13,552	1,177
Casa Tipo 2	11,961	1,546

Este experimento apresenta uma variação pequena dos tempos se comparado ao primeiro, o que leva à conclusão de que os testes necessários à identificação do analisador adequado de uma estrutura é de custo relativamente baixo.

8.2.3 Experimento III

Para o último experimento foi projetada uma região contendo uma estrutura correspondente a cada gramática de exemplo apresentada no Capítulo 6. O *bot* é posicionado no centro dessa região e efetua uma rotina de varredura do espaço em busca de estruturas aceitas por seus analisadores espaciais. Em cada posição é executada uma

análise a partir da gramática de segmento (6.6); se o resultado for positivo, então os analisadores das demais gramáticas serão utilizados para identificar se a posição em questão é a posição inicial de uma estrutura descrita pela gramática do analisador da vez. A medida de tempo considerada no Experimento III é o tempo que leva toda a varredura da região. A complexidade desse procedimento é semelhante à do experimento anterior, sendo incluída apenas a quantidade *q* de posições de leitura existentes na região analisada, ficando assim com a complexidade $\Theta(q\ t\ n\ r\ p\ \Theta(a))$; onde os demais termos têm a mesma representação do que no Experimento I.

Tabela 8-4 Resultados do Experimento III

Tamanho da região	Posições	Média (ms)	Desvio Padrão (ms)
10x10x10	1.000	454,2	23,848
15x15x15	3.375	903,0	65,824
20x20x20	8.000	1.123,5	82,995
30x30x30	27.000	1.711,7	108,165
50x50x50	125.000	4.885,1	500,854

Este último experimento demonstra que a varredura do espaço tem um impacto maior no resultado do que as análises léxicas e sintáticas isoladamente. Isso ocorre pelo grande número de posições que foram analisadas em cada caso e, para cada posição, a necessidade de verificar com cada analisador de cada gramática. É o acúmulo dessas tentativas com resultado negativo que explica esse grande aumento nos tempos.

Capítulo 9 - Conclusão

Este capítulo provê a conclusão deste trabalho, incluindo discussão dos resultados e destaques das contribuições da pesquisa. As sugestões de trabalhos futuros oferecem indicação de melhorias e expansões viáveis a partir dos resultados produzidos.

9.1 Epílogo

Esta dissertação demonstrou que é possível utilizar gramáticas posicionais livres de contexto para realizar a leitura e análise de espaços de *voxels*. Essa verificação se deu no ambiente do jogo eletrônico Minecraft, tendo como objetos de análise algumas das estruturas mais comumente encontradas pelas paisagens do jogo. Para atingir tal objetivo, foi necessário criar um analisador léxico espacial, relações posicionais e gramáticas específicas por classe de objeto analisado. Espera-se que o produto deste trabalho auxilie o desenvolvimento de *bots* no Minecraft capazes de considerar estruturas próximas para tomar decisões e locomover-se eficientemente pelo terreno.

9.2 Contribuições

As principais contribuições deste trabalho são:

- Organização e resumo dos trabalhos relevantes nas áreas de gramáticas de mais de uma dimensão e IAs em jogos e no Minecraft;
- Verificação da viabilidade do funcionamento de análise por gramáticas posicionais livres de contexto em três dimensões;
- O analisador léxico espacial, capaz de caminhar a posição atual de leitura sobre a entrada por meio de relações posicionais adequadas a cada estado, fazendo assim uma caminho linear de leitura pelo espaço multidimensional ocupado pelo objeto analisado;
- As gramáticas posicionais livres de contexto dos objetos analisados;
- As relações posicionais básicas e as específicas utilizadas na construção das gramáticas;

- Demonstração da viabilidade de analisar e aceitar estruturas com tamanhos variáveis em mais de uma dimensão;
- Possibilidade de que *bots* de Minecraft reconheçam estruturas de blocos em seus arredores.

9.3 Discussão dos resultados

Vale notar que o fato de o analisador léxico não ter acesso ao estado da análise sintática pode levar a uma escolha equivocada da próxima posição de leitura, o que resultará em uma possível rejeição de uma entrada válida. Isto ocorrerá sempre que uma relação posicional for viável num dado momento, segundo os termos da Definição 3-10, porém levar a redução de uma regra correspondente a uma relação posicional diferente da que foi executada pelo analisador léxico. Para contornar esse problema, o analisador léxico poderia ter acesso, durante a etapa de busca pela nova posição de leitura, à tabela LR e ao estado atual do analisador sintático. Dessa forma, as relações posicionais possíveis em cada momento estarão indicadas na tabela.

Como indicado nos experimentos, a performance das análises léxica e sintática são satisfatórias se fornecida uma posição inicial correta. O problema está em varrer o espaço de forma eficiente à procura de bons candidatos às análises gramaticais. Uma possibilidade de amenizar a questão é considerar que a cada movimento do *bot* não se faz necessário efetuar novamente uma varredura completa do espaço à sua volta, e sim de uma fatia menor do espaço, correspondente à região da qual o *bot* se aproximou em seu movimento. Isso torna o espaço a ser varrido menor e, portanto, torna a varredura incremental do espaço mais viável.

Para que os *bots* possam utilizar os analisadores de gramáticas posicionais para reconhecer objetos do espaço, é preciso que essas gramáticas tenham sido concebidas por humanos, pelo menos até o presente momento. Isso significa que estruturas encontradas pelo *bot* e não contempladas por nenhuma gramática serão ignoradas, o que pode representar uma desvantagem competitiva no jogo. Mais além, os analisadores das gramáticas apresentadas não são resistentes a erros, então um bloco fora do lugar invalida a estrutura na perspectiva do *bot*. Isso não seria um problema para um jogador humano.

Vale ainda ressaltar que, apesar de concebida e demostrada para dimensões superiores a 1 (um), as gramaticas posicionais livres de contexto podem ser usadas em leituras unidimensionais. Isso permite que o analisador visite novas posições de leitura

segundo as relações posicionais unidimensionais, não ficando atrelado à tradicional leitura da esquerda para a direita.

9.4 Trabalhos Futuros

A execução deste trabalho levou a concepção do analisador sintático espacial, entretanto, há ainda campo para aprimorar e complementar o funcionamento da análise. Estão listadas abaixo algumas sugestões de trabalhos futuros possíveis de serem desenvolvidos a partir dos resultados obtidos nesta pesquisa.

9.4.1 Substituir Heurística

A heurística usada no Algoritmo 5-1 baseia-se corretamente na Definição 3-10, mas assume o risco de escolher uma nova posição de leitura através de um operador posicional que não corresponde àquele momento da leitura na análise sintática. Substituir essa heurística e dar ao analisador léxico espacial o acesso à tabela LR(1) e ao estado atual do analisador sintático lhe dará condição de varrer o espaço em busca da próxima posição de leitura, considerando apenas as posições realmente viáveis àquele momento.

9.4.2 Leitura Prévia

Uma das capacidades mais úteis nos interpretadores e compiladores de linguagens amplamente usados é o *lookahead* (AHO; AHO, 2007). Essa técnica consiste em efetuar a leitura dos símbolos à frente antes que algumas decisões sejam tomadas ao longo da análise sintática. Assim eliminam-se algumas ambiguidades e adicionalmente permite-se que as linguagens fiquem mais enxutas, exatamente por contarem com formações que seriam ambíguas sem o uso do *lookahead*.

9.4.3 Recuperação de Erros

Atualmente, na solução proposta, a ausência de um bloco ou um bloco de tipo errado em qualquer posição causará um erro e consequentemente a rejeição da estrutura de entrada. Essa inflexibilidade, ou intolerância a erros, afeta consideravelmente a aplicação da solução em cenários reais, onde os erros são frequentes. (TOMITA, 1987) aponta um caminho viável ao desenvolvimento da recuperação de erros nos termos

tratados neste trabalho. Dispor da capacidade de avaliação de leitura descrita acima pode contribuir largamente para as tomadas de decisão necessárias à recuperação de erros.

9.4.4 Geração de Estruturas

Atualmente a geração de estruturas no Minecraft é feita por meio de algoritmos específicos por estrutura. As decisões sobre tamanhos e posições tomadas ao longo da geração são embasadas em números gerados aleatoriamente e tratados por tabelas de probabilidades. Entretanto, as gramáticas, por descrevem as regras de produção de corpos de uma linguagem, apresentam-se como ferramenta ideal para geração das estruturas de que os analisadores deste trabalho tratam. Apesar disso, no caso de gramáticas posicionais, há ainda que se decidir que posições candidatas devem ser escolhidas a cada avaliação de um operador posicional.

9.4.5 Descoberta de Gramáticas

Para que a análise de estruturas seja feita nos moldes deste trabalho, é necessária a concepção da gramática e das relações posicionais. Ter a capacidade de deduzir esses elementos a partir de exemplares de estruturas fornece uma enorme flexibilidade a quem faz uso dos analisadores gramaticais, pois lhe permite aprender novas gramáticas ao longo de sua experiência e, assim, estar mais preparado para os cenários que pode encontrar em seu futuro.

9.4.6 Otimização da Varredura

Diversas técnicas ainda podem ser empregadas com o objetivo de melhorar a performance da varredura do espaço, como a divisão da área de varredura em regiões menores e a paralelização das análises gramaticais. Outra possibilidade seria restringir ou priorizar a leitura de estruturas começadas a partir do chão ao invés de varrer linha a linha do espaço mesmo que este seja majoritariamente vazio.

Referências

AABY, A. A. Compiler construction using flex and bison. **Walla Walla College**, 2003.

ABEL, D. et al. **Goal-based action priors**Twenty-Fifth International Conference on Automated Planning and Scheduling. **Anais**...2015Disponível em: <http://www.aaai.org/ocs/index.php/ICAPS/ICAPS15/paper/view/10589>. Acesso em: 10 jun. 2015

ABEL, D.; TELLEX, S. Learning to Plan in Complex Stochastic Domains. 2015.

ADOBBATI, R. et al. **Gamebots: A 3d virtual world test-bed for multi-agent research**Proceedings of the second international workshop on Infrastructure for Agents, MAS, and Scalable MAS. **Anais**...Montreal, Canada, 2001Disponível em: <http://cs.biu.ac.il/~galk/Publications/01/gamebots.pdf>. Acesso em: 3 jun. 2015

AHA, D. W.; MUÑOZ-AVILA, H.; VAN LENT, M. **Reasoning, Representation, and Learning in Computer Games**Workshop Committee. **Anais**...2005Disponível em: <http://www.researchgate.net/profile/Kim_Bruce/publication/2581642_Workshop_Committee/links/09e4151225461da541000000.pdf>. Acesso em: 3 jun. 2015

AHO, A. V.; AHO, A. V. (EDS.). **Compilers: principles, techniques, & tools**. 2nd ed ed. Boston: Pearson/Addison Wesley, 2007.

ANDERSON, E. F. Playing smart-artificial intelligence in computer games. 2003.

ARRABALES, R.; LEDEZMA, A.; SANCHIS, A. **Towards conscious-like behavior in computer game characters**IEEE Symposium on Computational Intelligence and Games, 2009. CIG 2009. **Anais**... In: IEEE SYMPOSIUM ON COMPUTATIONAL INTELLIGENCE AND GAMES, 2009. CIG 2009. set. 2009

BACHMANN, P. **Die analytische zahlentheorie**. [s.l.] Teubner, 1894. v. 2

BARTH-MARON, G. et al. **Affordances as transferable knowledge for planning agents**2014 AAAI Fall Symposium Series. **Anais**...2014Disponível em: <http://h2r.cs.brown.edu/wp-content/uploads/2014/12/barth-maron14.pdf>. Acesso em: 10 jun. 2015

BAUCKHAGE, C.; THURAU, C.; SAGERER, G. Learning Human-Like Opponent Behavior for Interactive Computer Games. In: MICHAELIS, B.; KRELL, G. (Eds.). . **Pattern Recognition**. Lecture Notes in Computer Science. [s.l.] Springer Berlin Heidelberg, 2003. p. 148–155.

BAUMGARTEN, R.; COLTON, S.; MORRIS, M. Combining AI Methods for Learning Bots in a Real-Time Strategy Game. **International Journal of Computer Games Technology**, v. 2009, p. 1–10, 2009.

BAYLISS, J. D. **Teaching game AI through Minecraft mods**Games Innovation Conference (IGIC), 2012 IEEE International. **Anais**... In: GAMES INNOVATION CONFERENCE (IGIC), 2012 IEEE INTERNATIONAL. set. 2012

BEAUMONT, R. **bot made with mineflayer which can do task**. Disponível em: <https://github.com/rom1504/rbot>. Acesso em: 21 maio. 2015.

BELONGIE, S.; MALIK, J.; PUZICHA, J. Shape matching and object recognition using shape contexts. **IEEE Transactions on Pattern Analysis and Machine Intelligence**, v. 24, n. 4, p. 509–522, abr. 2002.

BERGER, M.; COLE, M. **Geometry I**. [s.l.] Springer Science & Business Media, 1987.

BERSON, A.; OTHERS. **Client/server architecture**. [s.l.] McGraw-Hill New York, 1992. v. 2

BONE, D. **Can Deterministic Context Free Grammars Catch-up in Defining Current Programming Languages?**, 1 dez. 2014.

BOYLE, R. **Artificially Intelligent Gamer Bots Convince Judges They're More Human Than Humans**. Disponível em: <http://www.popsci.com/technology/article/2012-09/artificially-intelligent-gamer-bots-convince-judges-theyre-more-human-humans>. Acesso em: 3 jun. 2015.

BURO, M.; CHURCHILL, D. Real-time strategy game competitions. **AI Magazine**, v. 33, n. 3, p. 106, 2012.

CARTER, Z. **Bison in JavaScript**. Disponível em: <https://github.com/zaach/jison>. Acesso em: 14 maio. 2015.

CHANG, S.-K. Picture processing grammar and its applications. **Information Sciences**, v. 3, n. 2, p. 121–148, abr. 1971.

CHARLES, D. **Enhancing gameplay: Challenges for artificial intelligence in digital games**Proceedings of the 1st World Conference on Digital Games. **Anais**...2003Disponível em: <http://www.researchgate.net/profile/Darryl_Charles/publication/221217607_Enhancing_gameplay_challenges_for_artificial_intelligence_in_digital_games/links/02e7e52af5b85d1b15000000.pdf>. Acesso em: 3 jun. 2015

COLE, N.; LOUIS, S. J.; MILES, C. **Using a genetic algorithm to tune first-person shooter bots**Congress on Evolutionary Computation, 2004. CEC2004. **Anais**... In: CONGRESS ON EVOLUTIONARY COMPUTATION, 2004. CEC2004. jun. 2004

COSTAGLIOLA, G. et al. **Automatic parser generation for pictorial languages**, Proceedings 1993 IEEE Symposium on Visual Languages, 1993. **Anais**... In: , PROCEEDINGS 1993 IEEE SYMPOSIUM ON VISUAL LANGUAGES, 1993. ago. 1993

COSTAGLIOLA, G. et al. Positional Grammars: A Formalism for LR-Like Parsing of Visual Languages. In: MARRIOTT, K.; MEYER, B. (Eds.). . **Visual Language Theory**. [s.l.] Springer New York, 1998. p. 171–191.

COSTAGLIOLA, G. et al. **On the pLR Parsability of Visual Languages.**HCC. **Anais**...IEEE Computer Society, 2003Disponível em: <http://dblp.uni-trier.de/db/conf/vl/hcc2001.html#CostagliolaDFG01>

COSTAGLIOLA, G.; CHANG, S.-K. **DR parsers: a generalization of LR parsers**, Proceedings of the 1990 IEEE Workshop on Visual Languages, 1990. **Anais**... In: , PROCEEDINGS OF THE 1990 IEEE WORKSHOP ON VISUAL LANGUAGES, 1990. out. 1990

COSTAGLIOLA, G.; CHANG, S.-K. Using Linear Positional Grammars for the LR Parsing of 2-D Symbolic Languages. **Grammars**, v. 2, n. 1, p. 1–34, 1999.

COSTAGLIOLA, G.; DEUFEMIA, V.; POLESE, G. Visual language implementation through standard compiler–compiler techniques. **Journal of Visual Languages & Computing**, v. 18, n. 2, p. 165–226, abr. 2007.

COSTAGLIOLA, G.; POLESE, G. **Extended positional grammars**2000 IEEE International Symposium on Visual Languages, 2000. Proceedings. **Anais**... In: 2000 IEEE INTERNATIONAL SYMPOSIUM ON VISUAL LANGUAGES, 2000. PROCEEDINGS. 2000

COSTAGLIOLA, G.; TOMITA, M.; CHANG, S.-K. **A generalized parser for 2-d languages**Visual Languages, 1991., Proceedings. 1991 IEEE Workshop on. **Anais**...IEEE, 1991Disponível em: <http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=238845>. Acesso em: 2 maio. 2015

CRIMI, C. et al. **Relation grammars for modelling multi-dimensional structures**, Proceedings of the 1990 IEEE Workshop on Visual Languages, 1990. **Anais**... In: , PROCEEDINGS OF THE 1990 IEEE WORKSHOP ON VISUAL LANGUAGES, 1990. out. 1990

CRUMLEY, Z.; MARAIAS, P.; GAIN, J. **Voxel-Space Shape Grammars**, 2012.

DARRYL, C. **Biologically Inspired Artificial Intelligence for Computer Games**. [s.l.] IGI Global, 2007.

DE VRIES, M. Three-dimensional grammar. **Linguistics in the Netherlands**, v. 20, n. 1, p. 201–212, 2003.

DOYLE, J.; DEAN, T. Strategic Directions in Artificial Intelligence. **ACM Comput. Surv.**, v. 28, n. 4, p. 653–670, dez. 1996.

DUNCAN, S. C. Minecraft, Beyond Construction and Survival. **Well Played**, v. 1, n. 1, p. 1–22, jan. 2011.

EARLEY, J. An efficient context-free parsing algorithm. **Communications of the ACM**, v. 13, n. 2, p. 94–102, 1970.

ESPARCIA-ALCÁZAR, A. I. et al. **Controlling bots in a First Person Shooter game using genetic algorithms**2010 IEEE Congress on Evolutionary Computation (CEC).

Anais... In: 2010 IEEE CONGRESS ON EVOLUTIONARY COMPUTATION (CEC). jul. 2010

FERNANDEZ-ARES, A. et al. **Optimizing player behavior in a real-time strategy game using evolutionary algorithms**2011 IEEE Congress on Evolutionary Computation (CEC). **Anais**... In: 2011 IEEE CONGRESS ON EVOLUTIONARY COMPUTATION (CEC). jun. 2011

GIANVECCHIO, S. et al. **Battle of Botcraft: Fighting Bots in Online Games with Human Observational Proofs**Proceedings of the 16th ACM Conference on Computer and Communications Security. **Anais**...: CCS '09.New York, NY, USA: ACM, 2009Disponível em: <http://doi.acm.org/10.1145/1653662.1653694>. Acesso em: 29 maio. 2015

GIPS, J. A syntax-directed program that performs a three-dimensional perceptual task. **Pattern Recognition**, v. 6, n. 3–4, p. 189–199, dez. 1974.

GOLDBERG, D.; LARSSON, L. **Minecraft: The Unlikely Tale of Markus "Notch" Persson and the Game that Changed Everything**. New York, NY, USA: Seven Stories Press, 2013.

GOODMAN, D. **JavaScript Bible**. [s.l.] John Wiley & Sons, 2007.

GORMAN, B.; HUMPHRYS, M. Imitative learning of combat behaviours in first-person computer games. **Proceedings of CGAMES**, 2007.

HAGELBÄCK, J.; JOHANSSON, S. J. **Using Multi-agent Potential Fields in Real-time Strategy Games**Proceedings of the 7th International Joint Conference on Autonomous Agents and Multiagent Systems - Volume 2. **Anais**...: AAMAS '08.Richland, SC: International Foundation for Autonomous Agents and Multiagent Systems, 2008Disponível em: <http://dl.acm.org/citation.cfm?id=1402298.1402312>. Acesso em: 3 jun. 2015

HAGELBÄCK, J.; JOHANSSON, S. J. A Multiagent Potential Field-based Bot for Real-time Strategy Games. **Int. J. Comput. Games Technol.**, v. 2009, p. 4:1–4:10, jan. 2009.

HENDRIKX, M. et al. Procedural content generation for games: A survey. **ACM Transactions on Multimedia Computing, Communications, and Applications**, v. 9, n. 1, p. 1–22, 1 fev. 2013.

HINDRIKS, K. V. et al. Unreal Goal Bots. In: DIGNUM, F. (Ed.). . **Agents for Games and Simulations II**. Lecture Notes in Computer Science. [s.l.] Springer Berlin Heidelberg, 2011. p. 1–18.

HINGSTON, P. A Turing Test for Computer Game Bots. **IEEE Transactions on Computational Intelligence and AI in Games**, v. 1, n. 3, p. 169–186, set. 2009.

HINGSTON, P. **A new design for a Turing Test for Bots**2010 IEEE Symposium on Computational Intelligence and Games (CIG). **Anais**... In: 2010 IEEE SYMPOSIUM ON COMPUTATIONAL INTELLIGENCE AND GAMES (CIG). ago. 2010

HLADKY, S.; BULITKO, V. **An evaluation of models for predicting opponent positions in first-person shooter video games**Computational Intelligence and Games, 2008. CIG '08. IEEE Symposium On. **Anais**... In: COMPUTATIONAL INTELLIGENCE AND GAMES, 2008. CIG '08. IEEE SYMPOSIUM ON. dez. 2008

JACOB, C.; MAMMEN, S. V. Swarm grammars: growing dynamic structures in 3D agent spaces. **Digital Creativity**, v. 18, n. 1, p. 54–64, 1 mar. 2007.

JOHNSON, D.; WILES, J. **Computer Games With Intelligence.**FUZZ-IEEE. **Anais**...Citeseer, 2001Disponível em: <http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.84.1848&rep=rep1&type=pdf>. Acesso em: 3 jun. 2015

KAMINKA, G. A. et al. GameBots: A Flexible Test Bed for Multiagent Team Research. **Commun. ACM**, v. 45, n. 1, p. 43–45, jan. 2002.

KELLEY, A. **Mineflayer plugin which gives bots a high level 3d navigating API using A***. Disponível em: <https://github.com/andrewrk/mineflayer-navigate>. Acesso em: 11 jun. 2015a.

KELLEY, A. **Mineflayer plugin to build or break blocks to get to a certain point**. Disponível em: <https://github.com/andrewrk/mineflayer-scaffold>. Acesso em: 11 jun. 2015b.

KELLEY, A. **A plugin to give you a web-based radar interface to your mineflayer bot**. Disponível em: <https://github.com/andrewrk/mineflayer-radar>. Acesso em: 11 jun. 2015c.

KELLEY, A. **A server that spins up mineflayer bots**. Disponível em: <https://github.com/andrewrk/mc-bot-server>. Acesso em: 11 jun. 2015d.

KELLEY, A. **Mineflayer bot that engages you in a gentlemanly duel**. Disponível em: <https://github.com/andrewrk/archerbot>. Acesso em: 11 jun. 2015e.

KELLEY, A. et al. **Create Minecraft bots with a powerful, stable, and high level JavaScript API**. Disponível em: <https://github.com/andrewrk/mineflayer>. Acesso em: 14 maio. 2015.

KHOO, A. et al. **Efficient, realistic NPC control systems using behavior-based techniques**AAAI Spring Symposium on Artificial Intelligence and Computer Games. **Anais**...2002Disponível em: <http://www.aaai.org/Papers/Symposia/Spring/2002/SS-02-01/SS02-01-010.pdf>. Acesso em: 3 jun. 2015

KHOO, A.; ZUBEK, R. Applying inexpensive AI techniques to computer games. **IEEE Intelligent Systems**, v. 17, n. 4, p. 48–53, jul. 2002.

KITANO, H. et al. The Robot World Cup Initiative. 1995.

KITANO, H. et al. The RoboCup synthetic agent challenge 97. In: **RoboCup-97: Robot Soccer World Cup I**. [s.l.] Springer, 1998. p. 62–73.

KNUTH, D. E. Big omicron and big omega and big theta. **ACM Sigact News**, v. 8, n. 2, p. 18–24, 1976.

KRISHNAMURTI, R. **Spatial Grammars: Motivation, Comparison, and New Results**, 2013.

KUBALL, C. **Mineflayer plugin which gives bots a function to find the nearest block**. Disponível em: <https://github.com/Darthfett/mineflayer-blockfinder>. Acesso em: 12 jun. 2015a.

KUBALL, C. **A mineflayer bot that helps you do things in minecraft.** Disponível em: <https://github.com/Darthfett/helperbot>. Acesso em: 21 maio. 2015b.

LAIRD, J. E. **It Knows What You'Re Going to Do: Adding Anticipation to a Quakebot**Proceedings of the Fifth International Conference on Autonomous Agents. **Anais**...: AGENTS '01.New York, NY, USA: ACM, 2001Disponível em: <http://doi.acm.org/10.1145/375735.376343>. Acesso em: 3 jun. 2015

LAIRD, J. E. Research in Human-level AI Using Computer Games. **Commun. ACM**, v. 45, n. 1, p. 32–35, jan. 2002.

LAIRD, J. E.; DUCHI, J. C. Creating human-like synthetic characters with multiple skill levels: A case study using the soar quakebot. **Ann Arbor**, v. 1001, p. 48109–2110, 2000.

LANDAU, E. **Handbuch der Lehre von der Verteilung der Primazahlen**. [s.l.] BG Teubner, 1909. v. 1

LE HY, R. et al. Teaching Bayesian behaviours to video game characters. **Robotics and Autonomous Systems**, Robot Learning from Demonstration. v. 47, n. 2–3, p. 177–185, 30 jun. 2004.

LIM, C.-U.; BAUMGARTEN, R.; COLTON, S. Evolving Behaviour Trees for the Commercial Game DEFCON. In: CHIO, C. D. et al. (Eds.). . **Applications of Evolutionary Computation**. Lecture Notes in Computer Science. [s.l.] Springer Berlin Heidelberg, 2010. p. 100–110.

LINDENMAYER, A. Developmental systems without cellular interactions, their languages and grammars. **Journal of Theoretical Biology**, v. 30, n. 3, p. 455–484, mar. 1971.

LIN, W. C.; FU, K. S. A Syntactic Approach to 3-D Object Representation. **IEEE Transactions on Pattern Analysis and Machine Intelligence**, v. 6, n. 3, p. 351–364, 1984.

LOWE, D. G. Three-dimensional object recognition from single two-dimensional images. **Artificial Intelligence**, v. 31, n. 3, p. 355–395, mar. 1987.

MACLEAN, S.; LABAHN, G. A new approach for recognizing handwritten mathematics using relational grammars and fuzzy sets. **International Journal on Document Analysis and Recognition (IJDAR)**, v. 16, n. 2, p. 139–163, jun. 2013.

MARTINOVIC, A.; VAN GOOL, L. Earley parsing for 2D stochastic context free grammars. **Technical report KUL/ESAT/PSI/1301, KU Leuven, ESAT, Leuven, Belgium**, 2013a.

MARTINOVIC, A.; VAN GOOL, L. **Bayesian grammar learning for inverse procedural modeling**Computer Vision and Pattern Recognition (CVPR), 2013 IEEE Conference on. **Anais**...IEEE, 2013bDisponível em: <http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=6618877>. Acesso em: 2 maio. 2015

MAURER, H. A.; SALOMAA, A.; WOOD, D. Pure grammars. **Information and Control**, v. 44, n. 1, p. 47–72, jan. 1980.

MCPARTLAND, M.; GALLAGHER, M. **Learning to be a Bot: Reinforcement Learning in Shooter Games.**AIIDE. **Anais**...2008aDisponível em: <http://www.aaai.org/Papers/AIIDE/2008/AIIDE08-013.pdf>. Acesso em: 3 jun. 2015

MCPARTLAND, M.; GALLAGHER, M. **Creating a multi-purpose first person shooter bot with reinforcement learning**Computational Intelligence and Games, 2008. CIG '08. IEEE Symposium On. **Anais**... In: COMPUTATIONAL INTELLIGENCE AND GAMES, 2008. CIG '08. IEEE SYMPOSIUM ON. dez. 2008b

MCPARTLAND, M.; GALLAGHER, M. Reinforcement Learning in First Person Shooter Games. **IEEE Transactions on Computational Intelligence and AI in Games**, v. 3, n. 1, p. 43–56, mar. 2011.

MCSTATS.ORG. **Minecraft Global Statistics**, 2015. Disponível em: <http://mcstats.org/global/>. Acesso em: 21 maio. 2015

MICROSOFT. **Minecraft to join Microsoft | News Center**, 2014. Disponível em: <http://news.microsoft.com/2014/09/15/minecraft-to-join-microsoft/>. Acesso em: 19 maio. 2015

MIIKKULAINEN, R. et al. Computational intelligence in games. **Computational Intelligence: Principles and Practice**, p. 155–191, 2006.

MOJANG. **Mojang — Makers of Minecraft**. Disponível em: <https://mojang.com/>. Acesso em: 19 maio. 2015.

MORA, A. M. et al. Evolving Bot AI in UnrealTM. In: CHIO, C. D. et al. (Eds.). . **Applications of Evolutionary Computation**. Lecture Notes in Computer Science. [s.l.] Springer Berlin Heidelberg, 2010. p. 171–180.

MUHAR, A. Three-dimensional modelling and visualisation of vegetation for landscape simulation. **Landscape and Urban Planning**, Our Visual Landscape: analysis, modeling, visualization and protection. v. 54, n. 1–4, p. 5–17, 25 maio 2001.

MUNOZ-AVILA, H.; FISHER, T. **Strategic planning for Unreal Tournament bots**AAAI Workshop on Challenges in Game AI. **Anais**...2004Disponível em: <http://www.aaai.org/Papers/Workshops/2004/WS-04-04/WS04-04-005.pdf>. Acesso em: 3 jun. 2015

NOTCH. **The home of Notch**, 2015. Disponível em: <http://notch.net/>. Acesso em: 19 maio. 2015

OWEN. **Yes, we're being bought by Microsoft**. Disponível em: <https://mojang.com/2014/09/yes-were-being-bought-by-microsoft/>. Acesso em: 19 maio. 2015.

PENG, K. J.; YAMAMOTO, T.; AOKI, Y. A new parsing scheme for plex grammars. **Pattern Recognition**, v. 23, n. 3–4, p. 393–402, 1990.

PETR JIRICKA, J. K. Deterministic forgetting planar automata are more powerful than non-deterministic finite-state planar automata. p. 71–80, 1999.

PIAZZALUNGA, U.; FITZHORN, P. Note on a three-dimensional shape grammar interpreter. **Environment And Planning B**, v. 25, p. 11–30, 1998.

PICKEM, D. **3D reconfiguration using graph grammars for modular robotics**. Thesis—[s.l.] Georgia Institute of Technology, 16 dez. 2011.

PRIESTERJAHN, S. et al. **Evolution of Human-Competitive Agents in Modern Computer Games**IEEE Congress on Evolutionary Computation, 2006. CEC 2006. **Anais**... In: IEEE CONGRESS ON EVOLUTIONARY COMPUTATION, 2006. CEC 2006. 2006

PRUŠA, D. Two-dimensional context-free grammars. **ITAT**, v. 2001, p. 27–40, 2001.

RICH, E. **Artificial Intelligence**. New York, NY, USA: McGraw-Hill, Inc., 1983.

RIESENHUBER, M.; POGGIO, T. Hierarchical models of object recognition in cortex. **Nature Neuroscience**, v. 2, n. 11, p. 1019–1025, nov. 1999.

ROZENBERG, G. (ED.). **Handbook of Graph Grammars and Computing by Graph Transformation: Volume I. Foundations**. River Edge, NJ, USA: World Scientific Publishing Co., Inc., 1997.

SANTOS, L. **leonardosantos/spatial-jison-lex**. Disponível em: <https://github.com/leonardosantos/spatial-jison-lex>. Acesso em: 14 maio. 2015.

SCHADD, F.; BAKKES, S.; SPRONCK, P. **Opponent Modeling in Real-Time Strategy Games.**GAMEON. **Anais**...2007Disponível em: <http://mzrtaiengine.googlecode.com/svn/trunk/Papers%20and%20Researches/RTS%20Games/Adaptation,Learning%20and%20Opponent%20Modelling/Opponent%20Modelling/Oponnent%20modelling%20in%20real-time%20strategy%20games%20-%20after%202007.pdf>. Acesso em: 3 jun. 2015

SCHAEFFER, J.; BULITKO, V.; BURO, M. Bots Get Smart. **IEEE Spectrum**, v. 45, n. 12, p. 48–56, dez. 2008.

SCHRUM, J.; KARPOV, I. V.; MIIKKULAINEN, R. **UT^2 Human-like behavior via neuroevolution of combat behavior and replay of human traces**2011 IEEE Conference on Computational Intelligence and Games (CIG). **Anais**... In: 2011 IEEE

CONFERENCE ON COMPUTATIONAL INTELLIGENCE AND GAMES (CIG). ago. 2011

SILVERMAN, B. G. et al. Human Behavior Models for Agents in Simulators and Games: Part I: Enabling Science with PMFserv. **Presence: Teleoper. Virtual Environ.**, v. 15, n. 2, p. 139–162, abr. 2006a.

SILVERMAN, B. G. et al. Human Behavior Models for Agents in Simulators and Games: Part II: Gamebot Engineering with PMFserv. **Presence: Teleoperators and Virtual Environments**, v. 15, n. 2, p. 163–185, 1 abr. 2006b.

SIROMONEY, R.; KRITHIVASAN, K.; SIROMONEY, G. n-Dimensional array languages and description of crystal Symmetry—II. **Proceedings of the Indian Academy of Sciences - Section A**, v. 78, n. 3, p. 130–139, 1 set. 1973.

SMITH, A. R. **A Pixel Is Not A Little Square, A Pixel Is Not A Little Square, A Pixel Is Not A Little Square! (And a Voxel is Not a Little Cube**. [s.l.] Technical Memo 6, Microsoft Research, 1995.

SOCHER, R. et al. **Parsing With Compositional Vector Grammars**In Proceedings of the ACL conference. **Anais**...2013

SONI, B.; HINGSTON, P. **Bots trained to play like a human are more fun**IEEE International Joint Conference on Neural Networks, 2008. IJCNN 2008. (IEEE World Congress on Computational Intelligence). **Anais**... In: IEEE INTERNATIONAL JOINT CONFERENCE ON NEURAL NETWORKS, 2008. IJCNN 2008. (IEEE WORLD CONGRESS ON COMPUTATIONAL INTELLIGENCE). jun. 2008

SPORTELLI, F.; TOTO, G.; VESSIO, G. A Probabilistic Grammar for Procedural Content Generation. 2014.

SPRONCK, P.; SPRINKHUIZEN-KUYPER, I.; POSTMA, E. Improving opponent intelligence through offline evolutionary learning. **International Journal of Intelligent Games and Simulation**, v. 2, n. 1, p. 20–27, 2003.

STACEY, K. **Using Minecraft to unboggle the robot mind**. Disponível em: <https://news.brown.edu/articles/2015/06/minecraft>. Acesso em: 10 jun. 2015.

STEEN, G. The Paradox of Metaphor: Why We Need a Three-Dimensional Model of Metaphor. **Metaphor and Symbol**, v. 23, n. 4, p. 213–241, 21 out. 2008.

STINY, G.; GIPS, J. **Shape Grammars and the Generative Specification of Painting and Sculpture.**IFIP Congress (2). **Anais**...1971Disponível em: <http://www.academia.edu/download/31016464/SGBestPapers72.pdf>. Acesso em: 3 jun. 2015

STOLCKE, A. **Bayesian Learning of Probabilistic Language Models**. [s.l: s.n.].

STORM, D. **Minecraft thin client and automation framework**. Disponível em: <https://github.com/DarkStorm652/DarkBot>. Acesso em: 21 maio. 2015.

SUBRAMANIAN, K. G. et al. **Two-dimensional picture grammar models**Computer Modeling and Simulation, 2008. EMS'08. Second UKSIM European Symposium on. **Anais**...IEEE, 2008Disponível em: <http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=4625283>. Acesso em: 2 maio. 2015

SUBRAMANIAN, K. G. et al. Pure 2D picture grammars and languages. **Discrete Applied Mathematics**, v. 157, n. 16, p. 3401–3411, ago. 2009.

TEBOUL, O. **Shape Grammar Parsing: Application to Image-based Modeling**. [s.l.] Ecole Centrale Paris, 2011.

THURAU, C.; BAUCKHAGE, C.; SAGERER, G. **Imitation learning at all levels of game-AI**Proceedings of the international conference on computer games, artificial intelligence, design and education. **Anais**...2004Disponível em: <http://www.researchgate.net/profile/Christian_Thurau2/publication/228474437_Imitation_learning_at_all_levels_of_game-AI/links/09e4151290bcc65898000000.pdf>. Acesso em: 29 maio. 2015

TILKOV, S.; VINOSKI, S. Node.js: Using JavaScript to Build High-Performance Network Programs. **IEEE Internet Computing**, v. 14, n. 6, p. 80–83, 2010.

TOMITA, M. An efficient augmented-context-free parsing algorithm. **Computational linguistics**, v. 13, n. 1-2, p. 31–46, 1987.

TOMITA, M. Parsing 2-Dimensional Language. In: TOMITA, M. (Ed.). . **Current Issues in Parsing Technology**. The Springer International Series in Engineering and Computer Science. [s.l.] Springer US, 1991. p. 277–289.

VAN LENT, M.; LAIRD, J. Developing an artificial intelligence engine. **Ann Arbor**, v. 1001, p. 48109–2110, 1998.

VON AHN, L.; LIU, R.; BLUM, M. **Peekaboom: A Game for Locating Objects in Images**Proceedings of the SIGCHI Conference on Human Factors in Computing Systems. **Anais**...: CHI '06.New York, NY, USA: ACM, 2006Disponível em: <http://doi.acm.org/10.1145/1124772.1124782>. Acesso em: 29 maio. 2015

WANG, D. et al. **Creating human-like autonomous players in real-time first person shooter computer games**Proceedings, Twenty-First Annual Conference on Innovative Applications of Artificial Intelligence. **Anais**...2009Disponível em: <https://dawn.ynet.sk/~luvar/skola/clanky/Creating%20Human-like%20Autonomous%20Players%20IA%202009.pdf>. Acesso em: 3 jun. 2015

WANG, P. S. P. **Three-dimensional object representation by array grammars**1991Disponível em: <http://dx.doi.org/10.1117/12.25152>. Acesso em: 16 jun. 2015

WITTENBURG, K. B.; WEITZMAN, L. M. Relational Grammars: Theory and Practice in a Visual Language Interface for Process Modeling. In: MARRIOTT, K.; MEYER, B. (Eds.). . **Visual Language Theory**. [s.l.] Springer New York, 1998. p. 193–217.

WOO, J.; KANG, A. R.; KIM, H. K. **Modeling of Bot Usage Diffusion Across Social Networks in MMORPGs**Proceedings of the Workshop at SIGGRAPH Asia. **Anais**...: WASA '12.New York, NY, USA: ACM, 2012Disponível em: <http://doi.acm.org/10.1145/2425296.2425299>. Acesso em: 29 maio. 2015

WURZER, G.; MARTENS, B.; BÜHLER, K. **3D Regular Expressions-Searching Shapes IN Meshes**eCAADe 2013: Computation and Performance–Proceedings of the 31st International Conference on Education and research in Computer Aided Architectural Design in Europe, Delft, The Netherlands, September 18-20, 2013. **Anais**...Faculty of Architecture, Delft University of Technology; eCAADe (Education and research in Computer Aided Architectural Design in Europe), 2013Disponível em: <http://repository.tudelft.nl/view/conferencepapers/uuid:250a72fb-f190-49e0-ae9e-3541d9f87852/>. Acesso em: 11 maio. 2015

YAMPOLSKIY, R. V.; GOVINDARAJU, V. Embedded Noninteractive Continuous Bot Detection. **Comput. Entertain.**, v. 5, n. 4, p. 7:1–7:11, mar. 2008.

ZADEH, L. A. Fuzzy sets. **Information and Control**, v. 8, n. 3, p. 338–353, jun. 1965.

ZUBEK, R.; KHOO, A. **Making the human care: On building engaging bots**AAAI Spring Symposium on Artificial Intelligence and Interactive Entertainment, Technical Report SS-02-01. AAAI Press, Menlo Park, CA. **Anais**...2002Disponível em: <http://www.aaai.org/Papers/Symposia/Spring/2002/SS-02-01/SS02-01-020.pdf>. Acesso em: 3 jun. 2015