

1 Tabuleiros

Foram criadas funções para criação, leitura e escrita de tabuleiros que serão usados no presente trabalho. Para a manipulação e visualização, mostrou-se mais fácil utilizar matrizes numéricas para representar os objetos sobre o tabuleiro, enquanto nos foi requisitado o uso de representação com caracteres para as casas na leitura e escrita dos tabuleiros. Desta forma, estabeleceu-se o código apresentado pela tabela 1, implementado no dicionário `str2n` contido no arquivo `utils.py`.

Significado	Código de caracteres	Código numérico
Casa livre	*	0
Parede ou obstáculo	-	4
Casa objetivo	\$	3
Casa de início da busca	#	2
Marcações temporárias	(Nenhum)	<1

Table 1: Tabela de conversão entre os diferentes códigos utilizados no projeto.

1.1 Criação dos tabuleiros

Criou-se algoritmo para a geração automática de tabuleiros, com suas paredes e os pontos de início e fim da busca. Visto que se deseja aparência semelhante à exposta na descrição do projeto (Figura ??), não seria possível a geração completamente aleatória das paredes, e fez-se necessário desenvolver a estratégia descrita adiante.

1.1.1 Visão geral

O processo de gerar tabuleiros é executado pelo script `gen_boards.py`, mais especificamente e em seu mais alto nível pela função `gen_board` implementada no referido *script*, acontecendo da seguinte maneira:

1. É criado um tabuleiro (matrix) em branco (preenchido com zeros), a partir das dimensões informadas com argumento;
2. São sorteados dois pontos aleatoriamente para servirem de início e fim da busca. São sorteadas quantas vezes forem preciso até que tenham distância Manhattan entre si maior que um comprimento heurísticamente definido como a soma das dimensões do tabuleiro sobre 2;
3. São construídas as paredes, como melhor explicado posteriormente.

[código da função]

1.1.2 Orquestração da construção das paredes

As peças mais importantes desse processo são as duas funções `build_walls` e `random_walk`, a primeira sendo de nível superior. O algoritmo de criação das paredes segue o seguinte raciocínio, coordenado pela `build_walls`:

1. Uma casa aleatória do tabuleiro (chamada *seed* no código) é sorteada por meio da função `seeds_gen`;
2. É desenhada uma parede a partir dessa casa com a função `random walk`;
3. O processo é repetido `nseeds` vezes, um dos parâmetros da função `build_walls`, definido empiricamente por padrão como um décimo da área do tabuleiro.

[código da função]

1.1.3 Construção de cada parede

A partir de cada semente (casa aleatória do tabuleiro) fornecida à função `random_walk` pela função `build_walls`, será traçada (ou pelo menos tentar-se-á traçar) uma nova parede. O papel da `random_walk` é então "andar" pelas casas do tabuleiro marcando-o com o símbolo que designará aqueles quadrados como um novo obstáculo. A `random_walk` pede como argumento duas funções essenciais, a `end_func` e a `turn_func`, que devem receber o comprimento do caminho traçado e retornar um valor booleano. O processo de traçado executado pela `random_walk` é então esclarecido a seguir:

1. A partir da posição inicial (argumento **start**), verifica-se quais são os deslocamentos unitários possíveis a partir **start**, isto é, que não levarão a casas ocupadas por algum obstáculo, que levarão a casas marcadas com 0, e sorteia-se um desses "passos" (tuplas no formato (1,0), (-1, 1), (0, -1), etc.). Os passos podem ser restritos aos ortogonais (baixo, cima, direita, esquerda) definindo como **True** o parâmetro **orth**;
2. Se não houver passo possível, a parede não é criada;
3. Caso contrário desloca-se a posição para posição + passo e marca-se essa casa como parede (o número marcado é dado pelo argumento **trail**). As outras casas do tabuleiro referentes aos outros passos possíveis não escolhidos são também marcadas com algum número menor que 1 (0.1 no caso), para que não sejam ocupadas em iterações posteriores e mantenham as paredes separadas entre si;
4. Esse processo de deslocamento e marcação prossegue, avançando com o mesmo passo sorteado, na mesma direção, até que:
 - (a) É encontrado um obstáculo (casa do tabuleiro com valor não nulo) à frente na direção escolhida atual;
 - (b) A função **turn_func** retorne **True**, caso em que a direção (passo) será sorteada novamente, ou;
 - (c) A função **end_func** retorne **True**, caso em que a criação da parede será finalizada.

As funções **end_func** e **turn_func** são uma boa forma de controlar a dinâmica da criação de paredes. Se esses argumentos da função **random_walk** são providos a ela como **floats** entre 0 e 1, a **random_walk** os substitui por funções que retornam **True** com a probabilidade representada pelos **floats** fornecidos.

Outra possibilidade criada, é fornecer um inteiro como argumento **len** para a **random_walk**, caso em que **end_func** se torna função que retorna **True** se a distância traçada for maior que o inteiro fornecido. Nesse caso, o inteiro representaria um comprimento máximo para a parede, de forma que ela seria finalizada por colisão com uma casa não vazia ou por atingir esse comprimento máximo.

Para os experimentos são usadas **turn_func = 0.2** e **end_func = 0**, de forma que há sempre um quinto de probabilidade de virar, e a parede será desenhada até que se encontre um obstáculo.

[Resultados paredes]

2 Algoritmos de busca

2.1 Busca cega

2.1.1 Busca em profundidade

2.1.2 Busca em largura

2.2 Busca informada

2.2.1 Busca *best-first*

Esse algoritmo de busca é implementado pela função

2.2.2 Busca A*