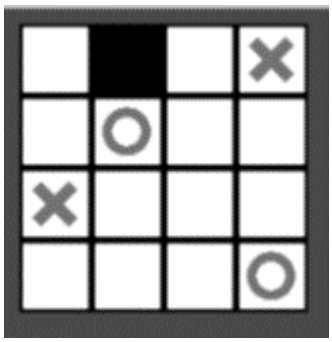
Universidade do Minho

Gerador de Mapas Aleatórios



Mestrado Integrado em Engenharia Informática

Grupo 073

Relatório da parte do
Gerador de Mapas Aleatórios

Este relatório tem como principal objetivo a descrição de um aglomerado de funções que devidamente entrelaçadas geram mapas/puzzles aleatórios de várias dificuldades (nomeadamente o inteiro 1 para representar um grau fácil e o inteiro 2 para representar um grau difícil).

A fim de encontrar uma forma de gerar mapas válidos com diferentes dificuldades, seguiu-se uma estrutura de raciocínio apoiada na construção por fases.

Primeiramente, fez-se a função **ESTADO vazia (int nL, int nC)** que percorre as linhas e as colunas e retorna um estado vazio.

De seguida a função **ESTADO daBloq (ESTADO e, int dif)** que retorna um estado atualizado com peças bloqueadas, sendo que a dificuldade foi implementada nesta função, visto que a linha de pensamento percorrida passou por: quantas mais peças bloqueadas menor a dificuldade, deste modo, quando o grau de dificuldade é fácil, o mapa fica preenchido com mais peças bloqueadas do que no grau difícil, de acordo (claramente) com a dimensão do puzzle. Nesta função, implementou-se uma dificuldade predefinida de 25% do total de peças serem bloqueadas, no caso do grau fácil e 12.5% do total das peças serem bloqueadas, se o grau de dificuldade for difícil.

```
if (dif == 1)
{
    numBloq = 0.25 * totalPecas;
}
else if (dif == 2)
{
    numBloq = 0.125 * totalPecas;
}
```

Assim, entramos num ciclo while que é percorrido enquanto o número de peças bloqueadas colocadas for menor que o número de peças bloqueadas que deveriam estar na grelha (segundo a dificuldade que esta possuir), vai gerando aleatoriamente linhas e colunas e se as peças forem vazias vai substituir por peças bloqueadas e incrementando esse valor (não havendo o risco de estarem a ser colocadas peças bloqueadas na mesma posição).

```
if (e.grelha[i][j] == VAZIA)
{
        e.grelha[i][j] = BLOQUEADA;
        bC++;
}
```

Depois, realizou-se a função **ESTADO daFixos (ESTADO e)** que retorna um estado atualizado com peças fixas, o método de raciocínio, neste caso, passou por definir que o número de peças fixas existentes num mapa seria dado pela média, ou seja, pela divisão por dois do somatório do número de linhas e colunas (int numFixos = (e.num_lins + e.num_cols) / 2;), o ciclo while assegura que o número de peças fixas vai sendo colocado enquanto esse mesmo número ainda não é o pretendido, vai gerando um número aleatório de linhas e colunas, no caso de a peça ser vazia, existe 50% de probabilidade de desenhar uma peça fixa X ou uma peça fixa O e vai incrementando o valor das peças fixas colocadas.

```
if (e.grelha[i][j] == VAZIA)
{
        e.grelha[i][j] = (rand() % 2 == 0) ? FIXO_X : FIXO_O;
        fC++;
}
```

Por fim, a função **ESTADO gerarP (int dif, int nL, int nC)** que desenha mapas e verifica a sua validade, e caso esta última não seja confirmada, a função continua a gerar mapas até serem válidos, para os poder apresentar (sendo esta parte representada pelo while).

```
while (!win(x) || !validaJogo(e) || !validaJogo(x))
{
    e = gerarP(dif, nL, nC);
    x = autoSolver(e);
}
```

Ao longo desta função é desenhado um mapa com peças vazias, bloqueadas e fixas, utilizando as funções acima referidas.

No código existem ainda funções das quais me servi para testar o funcionamento daquelas referidas anteriormente.

Em suma, penso que a estratégia adotada foi adequada e executada com sucesso.