3º Trabalho Prático

Programação

Realizado por:
<46055> João Martins
<46001> José Santos
<46074> Ricardo Margalhau

Instituto Superior de Engenharia de Lisboa – Inverno 2018 / 2019

1ª Funcionalidade

Não permitir colocar a peça nas posições da grelha onde não existe lugar para todas as molduras da peça

Para que o utilizador não possa colocar a peça num lugar onde esta não caiba, o nosso grupo criou um método na classe ***Board*** chamado ***hasSpaceForPiece***. Este método recebe como argumentos a peça que o utilizador está a tentar colocar e o lugar da *grid* que selecionou para tal, percorrendo cada *frame* da peça que o utilizador está a tentar colocar, caso o *frame* tenha cor (ou seja, é diferente de ***NO_FRAME***) e o mesmo *frame* na posição da *board* que o utilizador selecionou também tenha cor, então é impossível colocar a peça em questão.

O método ***hasSpaceForPiece*** é chamado mais tarde na classe ***ColorFrames*** no método ***putPieceInBoard***, método este que vai imprimir uma mensagem de erro caso a peça não caiba.

Caso se verifique que a peça cabe na *board* então adicionamos a peça a um *array* bidimensional com o nome de ***board*** na classe ***Board***, geramos e apresentamos a peça seguinte ao utilizador.

2ª Funcionalidade

Gerar apenas peças que possam ser colocadas numa ou mais posições da grelha.

Para verificar se a peça vai caber numa ou mais posições da *grid*, modificámos o método ***generatePiece*** da classe ***ColorFrames*** e criámos um método chamado ***isPieceAvailable*** na classe ***Board***. Este novo método tem como argumento a peça que está a ser gerada e vai percorrer cada posição da grelha verificando através do método ***hasSpaceForPiece*** (feito anteriormente) se a peça cabe na posição da board onde a variável de iteração do *for* está. Caso caiba retorna ***true*** e não chega a percorrer as restantes posições. Se, após o ciclo, não tenha retornado ***true*** então irá retornar ***false*** (não encontrou uma posição onde a peça cabe).

No método ***generatePiece*** optámos por gerar uma nova peça enquanto a chamada ao método ***isPieceAvailable*** retorne ***false***. Isto tratou-se de implementar um ***do-while*** pois a primeira vez irá ser incondicional. Dentro deste ciclo fazemos *reset* ao *array* de inteiros ***piece*** e usamos o *for* que já vinha feito para gerar a peça, e enquanto a peça não caiba vai continuar a fazer estes dois *for's*.

3ª Funcionalidade

Terminar o jogo quando a grelha estiver completamente preenchida de molduras.

Para gerar uma peça que possa ser colocada numa ou mais posições da grelha houve a necessidade de alterar o método ***generatePiece***. Ao alterar este método, tivemos de criar um método ***getMaxFramesPerSquare*** na classe ***Board***, sendo que este vai analisar quantos *frames* no máximo é que o ***generatePiece*** pode gerar. Caso o retorno deste método venha a zero, ou seja, já não podemos gerar mais *frames*, então estamos perante um *game over* e, sendo assim, afetamos a *flag* ***terminate*** da classe ***ColorFrames*** com o valor lógico ***true***. Caso o retorno venha maior que zero (não pode ser negativo pois trata-se do máximo de *frames*) então substituímos a *flag* ***FRAMES_DIM*** pelo retorno do método na variável ***numOfFrames*** que está dentro do método ***generatePiece*** (gerando assim o número máximo de *frames* possível).

4ª Funcionalidade

Detetar a formação de linhas, colunas, diagonais e células com molduras da mesma cor, fazendo-as desaparecer (sem a funcionalidade 6) e contabilizando a pontuação.

Para realizar esta funcionalidade o nosso grupo começou por verificar as molduras da mesma cor e para tal fizemos o método ***checkGridPos***, que recebe como argumento a posição para verificar na classe ***Board***, este método é chamado sempre que uma nova peça é adicionada à *board*. Para verificar as molduras bastou-nos criar um ciclo *for* dentro deste método que vai verificar por cada *frame* se a cor deste é diferente da cor do primeiro *frame* ou se a cor do *frame* iterado é igual a ***NO_FRAME*** e, caso esta condição venha a ser **falsa**, verificamos que o ciclo chegou ao seu fim, significando que a peça vai ser contabilizada. Caso o ciclo não tenha chegado ao fim (ou seja, a condição foi verdadeira numa das iterações), verifica a formação de colunas, linhas ou diagonais de cada *frame* da peça (um ciclo *for*).

Para verificar as linhas, colunas e diagonais começámos por criar um método "genérico" na classe ***Board*** com o nome de ***check*** que vai procurar pela cor começando na posição inicial ***pos*** diferentes da posição em que o utilizador inseriu a peça com a cor que o programa vai procurar (***ppos***), incrementando por cada iteração o valor ***pos*** com variável inteira ***mp*** multiplicada pela variável de iteração do primeiro ciclo, e por cada peça verificada, caso se verifique que a cor dos *frames* são todas diferentes da cor especificada (***color***) então termina ambos os ciclos iterativos com a instrução *break*. Caso contrário se verificar-se pelo menos um *frame* com a cor especificada em ***color*** nos espaços da *grid* separados por ***mp*** então apagamos estas peças através do método ***clearGridPositionColor*** que recebe como argumentos a posição e a cor para ser limpa.

Após o método ***check*** realizado usámos o mesmo dentro dos métodos ***checkLine***, ***checkColumns*** e ***checkDiagonals***.

No ***checkLine*** começámos por encontrar a posição inicial da linha que, por exemplo, numa *grid* 3x3 poderia ser 1, 4 ou 7, sendo que para isto usámos um ciclo ***while*** (a posição poderia já ser a inicial) e enquanto o resto da posição anterior com ***BOARD_DIM*** for maior que zero e a posição for maior que um então decrementamos a posição. Após isto feito usamos o método ***check*** passando os respetivos parâmetros, sendo que, neste caso, o argumento ***mp*** é um pois cada linha é um conjunto de valores separados por este inteiro.

No ***checkColumn*** começámos por encontrar a posição inicial da linha que, por exemplo, numa *grid* 3x3 poderia ser 1, 2 ou 3, sendo que para isto usámos um ciclo ***while*** (a posição poderia já ser a inicial) e enquanto a posição subtraindo com ***BOARD_DIM*** for maior que zero então decrementamos a posição com ***BOARD_DIM***. Após isto feito usamos o método ***check*** passando os respetivos parâmetros, sendo que, neste caso, o argumento ***mp*** é ***BOARD_DIM*** pois cada coluna é um conjunto de valores separados por este inteiro.

No ***checkDiagonals*** já não foi preciso realizar um ciclo ***while*** como nos outros dois porque neste caso já sabíamos que os primeiros elementos das diagonais eram um e ***BOARD_DIM***. Para a primeira diagonal calculámos o ***mp*** e chegámos à conclusão que os valores desta incrementavam ***BOARD_DIM + 1***. Na segunda concluímos que o valor de incrementação era ***BOARD_DIM – 1***.

5ª Funcionalidade

Evoluir de nível passando a gerar molduras com mais cores possíveis, de acordo com a tabela apresentada.

Nível	Pontuação	Max. de Cores
1	Até 25	4
2	25 .. 50	5
3	50 .. 100	6
4	100 .. 200	7
5	200 .. 400	8
6	Mais de 400	9

Tabela 1 – Pontuação para cada nível e máximo de cores correspondente

Com a tabela apresentada o nosso grupo rapidamente verificou que o máximo de cores que poderiam vir a ser geradas consoante o nível seria ***nível + 3***. Posto isto, começámos por criar a classe ***Scoreboard*** que contem uma referência ***levelScores*** para um *array* de inteiros que armazena o *score* máximo de cada nível por ordem crescente. Após isto criámos mais dois campos ***score*** e ***level*** dos tipos ***int*** e ***byte*** respectivamente (para o ***level*** usámos ***byte*** pois com byte podemos armazenar todos os valores de 0 a 255 e basta-nos de 1 a 6).

Depois destes campos criados começámos por criar o método ***addPoints*** que tem como argumento quantos pontos irão ser adicionados a ***score***. Neste método adicionamos os pontos com um efeito dinâmico e, por esta mesma razão, precisámos de usar um *for* para os adicionar. Cada vez que pontos são adicionados o método vai verificar, caso o utilizador já não esteja no nível 6, se o ***score*** é suficiente para mudar de nível e, caso seja, chama o método ***nextLevel*** da mesma classe.

Outro método que adicionámos nesta classe na fase de implementação em que nos encontramos foi o ***getMaxColors*** que apenas retorna o máximo de cores consoante o nível em que estamos. Para que este método tivesse efeito no jogo houve a necessidade de alterar novamente o ***generatePiece*** da classe ***ColorFrames*** substituindo a constante ***MAX_COLORS*** que se encontrava dentro do segundo ciclo *for* pelo retorno do método. Como a constante previamente eliminada já não era precisa então o nosso grupo decidiu eliminar a mesma da classe ***ColorFrames***.