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 ***pieces*** 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, 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***, 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*** e vai avançando até chegar ao fim, não contando a posição ***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 se verificar 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 subtraída com ***BOARD_DIM*** for maior que zero, 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*** respetivamente (para o ***level*** usámos ***byte*** pois com este tipo podemos armazenar todos os valores de 0 a 255 e basta-nos de 1 a 6).

Depois de criar estes campos, 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***.

Para usar o método ***addFrames*** modificámos a classe ***Board*** criando um campo privado com o nome ***pointsToAdd*** sendo que este campo do tipo inteiro vai contabilizar quantos frames foram eliminados no total e, para isto, basta incrementar sempre que há uma chamada ao método ***clearGridPositionFrame***. No final do método ***checkGridPos,*** após eliminar os quadrados, chamamos o método ***addFrames*** passando como parâmetro ***pointsToAdd***. Finalmente fazemos *reset* a ***pointsToAdd***.

6ª Funcionalidade

Apresentar de forma intermitente as molduras de cores iguais que formaram as linhas, as colunas, as diagonais ou a célula.

Para esta funcionalidade o nosso grupo fez um pequeno efeito que apaga os frames com *200ms* de *delay*. Para fazer isto acrescentámos no método ***clearGridPosistionFrame***

uma chamada ao método ***sleep*** da classe ***Console*** caso a cor especificada como argumento seja diferente da constante ***NO_FRAME*** da classe ***ColorFrames***.

7ª Funcionalidade

Dar a hipótese de ser iniciado um novo jogo.

De modo a realizar esta funcionalidade reparámos logo desde início que para além da flag ***terminate*** da classe ***ColorFrames*** seria preciso ter outra denominada de ***forceTerminate*** para aquando do utilizador premir a tecla *ESC* o jogo sair sem perguntar pela confirmação do utilizador.

Após isto feito começámos a criar o método ***continuePlaying***, que retorna um valor do tipo *boolean*, na classe ***Panel***. Este método quando é chamado limpa a parte de baixo da janela (parte esta onde está o *score*, a próxima peça e a mensagem atual) e pergunta ao utilizador se o jogo deve ser reiniciado ou não. Antes de ficar à espera do carácter inserido pelo utilizador, foi preciso limpar os caracteres inseridos anteriormente com o método ***clearAllChars*** da classe ***Console*** pois caso não fizéssemos isto o jogo saía automaticamente sem perguntar. Depois de receber a resposta do utilizador limpamos novamente a parte de baixo e retornamos um *boolean* que indica *true* caso o caracter inserido for um 'Y' (*case-insensitive*).

Para chamar este novo método criámos um outro na classe ***Scoreboard*** denominado de ***endGame***. Este método foi criado pelo nosso grupo pois iriamos precisar dele mais à frente para guardar os *scores* mais altos. Neste método ao chamar o ***continuePlaying*** e caso o retorno seja *true*, fazemos *reset* à *grid* inteira e retornamos *false* neste novo método, ou seja, o jogo não vai terminar. Caso contrário retorna *true*.

Na classe ***ColorFrames*** no método ***playGame*** construímos mais um ciclo *do-while* de modo a apenas acabar o jogo caso a *flag* ***forceTerminate*** venha a *true* ou caso ***terminate*** esteja a true e o utilizador responda que quer acabar quando lhe for perguntado. Neste último caso é preciso fazer *reset* à *flag* ***terminate*** ao reiniciar o jogo pois caso contrário estamos perante um ciclo que não chega a iniciar o jogo e apenas termina com a resposta negativa por parte do utilizador.

8ª Funcionalidade

Mostrar o tempo decorrido desde o início do jogo.

Para realizar esta funcionalidade usámos o método ***currentTimeMillis*** da classe ***System***, método este que retorna o tempo atual em milissegundos desde o *epoch*, no início do primeiro ciclo *do-while* do método ***playGame***, pois nós queríamos que o tempo fosse reposto caso o jogo reiniciasse. Após isto bastou-nos no segundo *do-while* (o ciclo do jogo) obter o tempo atual, que vem com uma separação de aproximadamente um segundo do anterior devido ao ***waitKeyPressed***, subtrair este com o tempo do início e dividir esta subtração por 1000, sendo que assim podemos obter o tempo desde o início do jogo em segundos. Para apresentar este tempo ao utilizador criámos um método na classe ***Panel*** que transforma de segundos para minutos e segundos, apresentado o resultado por baixo do *score*.

9ª Funcionalidade

Guardar a tabela das 10 melhores pontuações com o respetivo nome do jogador.

De modo a implementar esta funcionalidade decidimos primeiramente criar uma nova classe com o nome de ***Player***. Esta nova classe, que tem dois campos públicos (o nome do jogador e a sua pontuação) e um construtor que requer como argumentos os valores para os campos desta classe, servirá para criar uma instância da mesma (um objeto) de modo a armazenar as duas propriedades referidas anteriormente para todos os jogadores que obtenham uma pontuação elevada.

Para usar a nova classe foi decidido criar, dentro da classe ***Scoreboard***, uma referência para um *array* de elementos desta nova classe com a dimensão de 10, ou seja, para armazenar os 10 melhores jogadores.

Após isto o nosso grupo precisou de verificar no final de cada jogo se a pontuação do jogador seria para armazenar na tabela de pontuações e, para tal, criámos um novo método denominado ***hasNewHighScore*** que retorna um valor do tipo *int*, sendo este valor o *index* onde o novo jogador será posto na tabela, ou, caso retorne -1, o jogador não tem pontuação suficiente para entrar para a tabela de pontuações. Para encontrar o *índex* para este novo jogador (se for caso) usámos uma iteração do algoritmo *Insertion Sort* para o efeito.

Para finalizar modicámos o método ***endGame*** da classe ***Scoreboard*** anteriormente criado para verificar se o jogador atingiu uma nova pontuação que permita um lugar na tabela e caso isto se verifique adiciona o mesmo à tabela, posteriormente apresentado esta tabela de pontuações através do método ***printScoreboard*** da classe ***Panel*** que itera sobre todos os jogadores do *scoreboard* e apresenta-os, caso haja espaço suficiente, na parte de baixo da janela.

10ª Funcionalidade

Usar também cliques do rato para selecionar a posição para colocar a peça.

Para realizar esta funcionalidade começámos por criar o método ***processMouseEvent*** na classe ***ColorFrames*** do jogo. Este método tem como objetivo receber as coordenadas do rato quando o utilizador premir o mesmo, e transformar estas coordenadas para a posição da grelha, se for aplicável.
De modo a obter coordenadas fomos chamar o método ***getMouseEvent*** com o argumento ***MouseEvent.Click*** da classe ***Console***. Este argumento permitiu-nos apenas obter o evento aquando o utilizador prime o botão do lado esquerdo do rato. Após isto fomos verificar se o retorno desta chamada vinha a *null*, pois caso não houvesse *Mouse Event* no momento isto poderia acontecer.

Depois de obter a posição ***x*** e ***y*** do rato fomos iterar por cada coluna e por cada linha da *grid* e concluíamos que o mínimo do rato para uma coluna ***i*** seria ***i * (Panel.GRID_SIZE + 1) + 1*** e com isto obtemos também o máximo somando apenas ***Panel.GRID_SIZE – 1***. Ao repetir este processo para os pontos ***x*** e ***y*** conseguimos verificar se a posição do rato se encontra dentro de um dos quadrados da grelha. Caso se encontre então vamos adicionar a peça ao quadrado ***i + j + (BOARD_PLACES - BOARD_DIM + 1) - (BOARD_DIM + 1) * j***, sendo que ***i*** é a iteração por cada coluna e ***j*** por cada linha.

Concluindo esta parte o nosso grupo ficou a saber que é possível simplificar ainda mais este processo, contudo até à data de entrega ainda não conseguimos encontrar um método mais simples para realizar esta operação.

Funcionalidade Adicional

Adicionar música no decorrer do jogo

De modo a tirarmos partido de todas as funcionalidades da biblioteca ***ConsolePG*** adicionámos a funcionalidade do jogo tocar música enquanto decorre. Para realizar tal foi preciso converter os ficheiros de música para o formato *wav* e salvar os mesmos para uma pasta com o nome de *sound* localizada na directoria de execução do programa.

Após todas as músicas adicionadas modificamos o método ***nextLevel*** da classe ***Scoreboard*** para que no ínicio do jogo comece a reproduzir a primeira música, e para que no último nível mude para uma música mais apropriada para o nível.