Instituto Superior de Engenharia de Lisboa

PROGRAMAÇÃO

Relatório do Terceiro Trabalho Prático

Docente

Profª. Matilde Pato

Alunos

44225 Rúben Rosário
44243 Rui Correia
43552 Samuel Costa

Janeiro 2017

Índice

Introdução

Este relatório, referente ao terceiro de três trabalhos práticos, foi elaborado no âmbito da unidade curricular de Programação e acompanha a implementação de uma aplicação para jogar ao 4 em linha. Este exercício constituiu uma aplicação dos conteúdos cobertos pelos dois primeiros trabalhos práticos bem como de outros aspetos da linguagem Java, designadamente, conceitos introdutórios à Programação Orientada para Objetos.

Ao longo do documento apresentam-se os principais objetivos metodológicos e práticos delineados pelo problema em questão; a metodologia adotada; uma explicação dos procedimentos aplicados no desenvolvimento da aplicação; e, por fim, uma breve reflexão sobre o trabalho realizado em que se discutem os resultados obtidos, explicitando-se, conclusões referentes à Programação Orientada para Objetos.

O documento inclui ainda uma adenda ("Anexo – Funcionalidades Opcionais"), que dá conta dos desenvolvimentos que o programa sofreu desde a entrega anterior, que incluía apenas as funcionalidades obrigatórias.

Objetivos

O trabalho prático visa, em termos gerais, aplicar as ferramentas introduzidas em contexto de sala de aula, em particular, os conteúdos introdutórios de Programação Orientada para Objetos, aprofundando as suas potencialidades de utilização, por meio do desenvolvimento de uma aplicação para jogar ao jogo "4 em linha".

Definem-se os seguintes objetivos:

- do ponto de vista metodológico: identificar os procedimentos a aplicar no desenvolvimento da aplicação;
- do ponto de vista prático, empregar e dominar os seguintes instrumentos da linguagem java: (i) respeitar a sintaxe do JAVA e as orientações de conceção de classes; (ii), reconhecer a diferença entre métodos de classe e de instância; (iii) reconhecer a diferença entre estado e comportamento de um objeto; (iv) desenvolver classes que possam ser referenciadas por tipo; e por último (v) efetuar passagem de objetos a métodos.

Métodos e Recursos Utilizados

Para a realização deste trabalho foi utilizada a linguagem *Java™ Versão 8 Update 111.* Todos os elementos da linguagem necessários para a resolução dos problemas propostos foram adquiridos nas aulas da unidade curricular Programação (PG).

Programação Estruturada *versus* Programação Orientada a Objectos

Para a realização deste trabalho foi necessário entender a diferença entre a programação estruturada (*Procedural paradigm*), até então aplicada nos dois primeiros trabalhos da unidade curricular, e a programação orientada a objectos (*Object-Oriented Paradigm*).

Nos dois primeiros trabalhos foram introduzidas e utilizadas técnicas de programação para revolver problemas, usando ciclos, estruturas de controle de fluxo, métodos, arrays e tipos primitivos, sendo estas técnicas a base tanto para a programa estruturada como para a programação orientada a objectos.

A programação estruturada caracteriza-se por se focar na conceção de métodos, enquanto que a programação orientada junta dados, métodos e objectos. Esta foca-se nos objectos e nas suas operações, usando também todas as vantagens da programação estruturada.

A programação orientada a objectos organiza o programa de uma forma similar ao mundo real, em que os objectos estão associados a atributos e a actividades, i.e., ao contrário da programação estruturada que coloca os dados e as operações separados, a programação orientada a objectos coloca as operações e os dados a que pertencem num objecto.

Em suma, um programa escrito sob o paradigma da programação orientada a objectos é um programa vais fácil de desenvolver e de mais fácil manutenção. Um programador não necessita de conhecer todo o código do programa para desenvolver parte dele, facilitando assim o trabalho em equipa para o seu desenvolvimento.

Classes e objectos

Uma classe é um conjunto de dados e métodos que operam sobre esses mesmos dados. Um objecto é uma instancia da classe, ou seja, pode-se criar várias instancias da classe (instanciar), criando assim vários objectos (ver figura 1).

Em Java, as classes usam variáveis para definir os atributos dessa classe e métodos para definir ações.

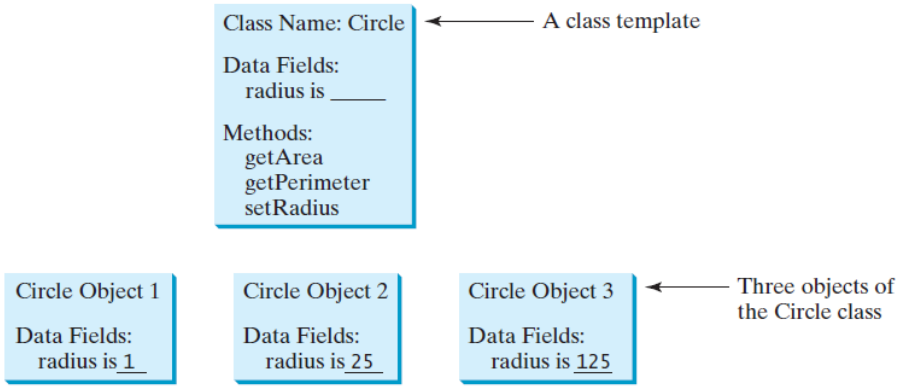


Figura 1 - Uma Classe é um "Template" para a criação de objectos.[3]

As classes podem ter dados e métodos acessíveis a outros objectos ou não, i.e., podem ser públicos ou privados. Por exemplo, uma classe deve conter métodos que providenciam ao cliente (por exemplo, outros objectos) o acesso a determinados dados ou ações sem os modificar (*private*), tornando-se assim uma classe de mais fácil manutenção.

Uma classe deve descrever uma única entidade, por exemplo, uma classe que defina estudantes não deve definir estudantes e professores na mesma classe (coesão).

Sempre que uma variável é partilhada por todas as instâncias de uma classe, esta deve ser declarada como estática.

ConsolePG como *input* e *output*

Para este trabalho foi utilizado a classe ConsolePG, que permite realizar escritas e leituras numa janela de texto. Esta permite escrever e ler em determinadas coordenadas (linhas e colunas) da janela, utilizando variadas cores. É possível definir cores tanto para os caracteres como para o fundo do texto (*background*).

Todas as funcionalidades desta classe podem ser consultadas num documento em formato Pdf fornecido para a realização deste trabalho.

Regras do Jogo

O 4 em linha é um jogo por turnos para dois jogadores. Os jogadores empilham peças da sua cor nas colunas de um tabuleiro, deslizando até à ultima posição livre, até um dos jogadores ter conseguido fazer uma sequência de 4 peças da sua cor na horizontal, vertical ou diagonal do tabuleiro antes do adversário.

Para este trabalho, foi requerido que o jogo pudesse ser alterado por forma a se poder jogar ao 3, 4 ou 5 em Linha. Esta alteração do jogo é realizada através da modificação do valor de um parâmetro, neste caso da variável "*N"* da classe *InLine* (exemplo: *N=3*, *N=4* ou *N=5*).

Procedimentos Aplicados

Para explicar como se pode jogar ao 4 em linha com o programa que foi desenvolvido, é útil traçar a distinção entre o jogo propriamente dito, que pode ser jogado num tabuleiro qualquer de 4 em linha, e o programa (este programa em particular). Esta distinção ajuda a tornar claro durante o texto, quando se falar do que foi feito para programar o jogo por um lado, e as características do jogo, que são as mesmas quer se esteja a jogar num tabuleiro físico ou num tabuleiro virtual no computador. Esta secção centra-se na programação do jogo, com a finalidade de cumprir com os requisitos de um programa para jogar ao 4 em linha, sendo de salientar que a programação não é independente do jogo propriamente dito.

Na figura 2 é apresentado um diagrama (baseado na linguagem *UML- Unified Modeling Language*) onde estão representadas as tês classes mais elevadas do programa desenvolvido, assim como os seus atributos e métodos.

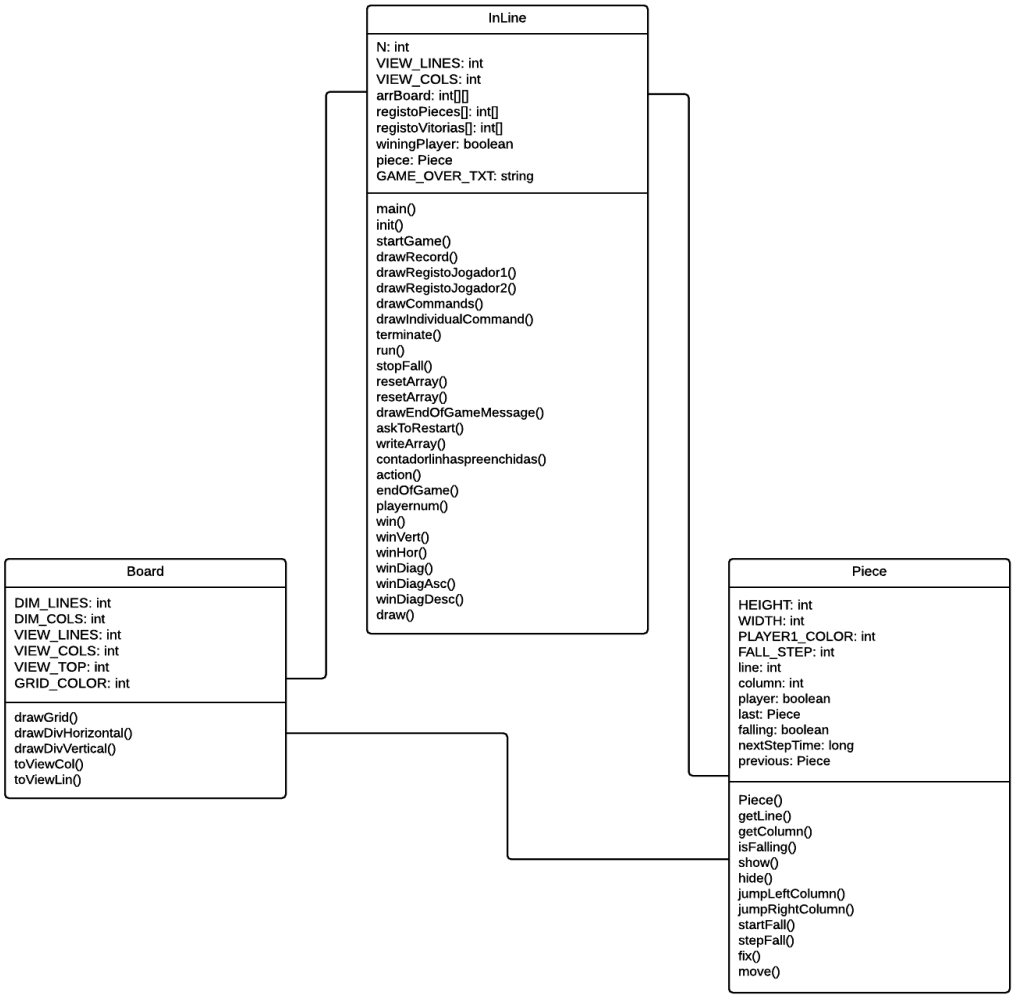
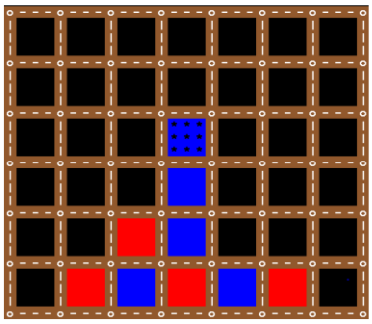


Figura 2 - Diagrama de classes do programa (UML).

Abstração do Tabuleiro em um *Array*

Para facilitar a verificação de posições livres, posições ocupadas e monitorizar o estado do jogo, recorreu-se a uma matriz bidimensional com as dimensões do tabuleiro. Como se pode visualizar na figura 3 o array (`arrBoard`) no inicio do jogo está totalmente preenchido com zeros, sendo que, à medida que os jogadores vão colocando as peças no tabuleiro, estes zeros são substituídos por “1” ou “2” consoante o jogador.



	0	1	2	3	4	5	6
0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0
2	0	0	0	1	0	0	0
3	0	0	0	1	0	0	0
4	0	0	2	1	0	0	0
5	0	2	1	2	1	2	0

Figura 3 - Abstração do tabuleiro de jogo em um array bidimensional.

Método *main()* e Método *run()*

Do ponto de vista da implementação do programa, o método `main()` encontra-se na classe `InLine`.

Assim que o programa é iniciado, este corre o método `main()`, que chama apenas dois métodos: o método `init()` e o método `run()` (ver Listagem 1).

```
public static void main(String[] args) {
    init();
    run();          // Loop of the game
}
```

Listagem 1 - Código do método "main".

O método `init()` tem como função abrir uma janela da consola com determinadas dimensões dependentes do jogo em questão (3 em Linha, 4 em Linha ou 5 em Linha), tendo esta como conteúdo pré-definido o tabuleiro vazio, e à direita do qual uma tabela de comandos do jogo.

No topo do tabuleiro encontra-se também uma peça do jogador ativo, sendo que, por defeito, o jogador 1 começa a jogar.

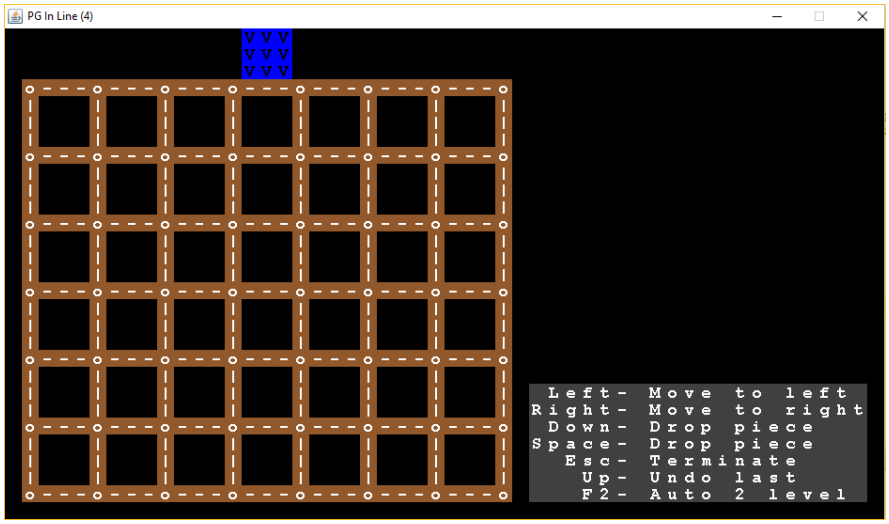


Figura 4 - Janela da consola criada pelo método init() onde é mostrado o tabuleiro vazio e uma tabela com os comandos de jogo.

O método `run()` cria um ciclo (*"loop of the game"*) em que vários métodos são chamados, permitindo assim os jogadores (utilizadores do programa) jogarem ao 4 em Linha. Ou seja, é o método `run()` que está a ser executado enquanto os jogadores estiverem a jogar.

Na figura 5 está representado o fluxograma do método `run()`:

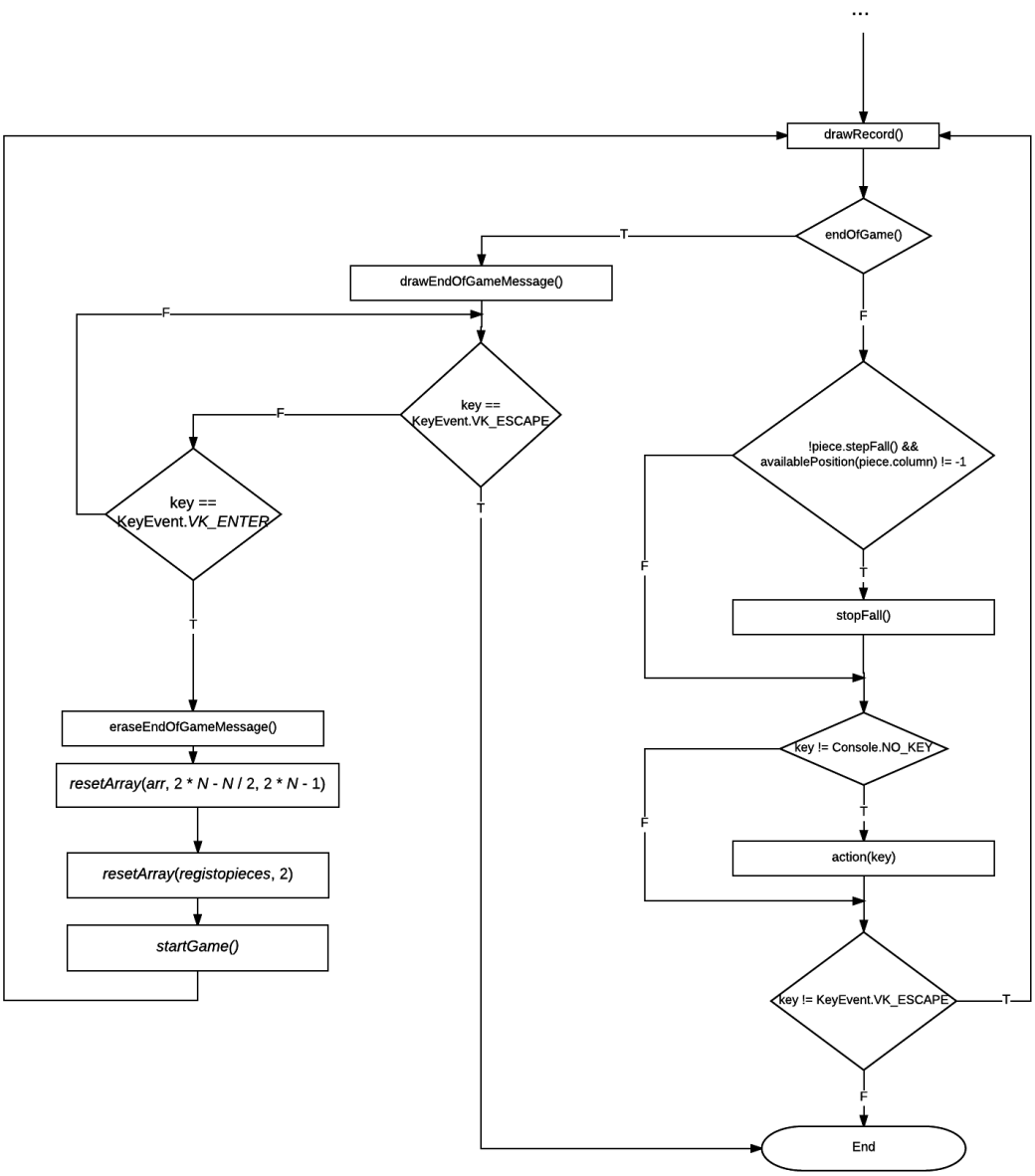


Figura 5 - Fluxograma do método run().

Para que o registo de vitórias e de número de peças jogadas por cada jogador seja atualizado enquanto os dois jogadores jogam, foi escrito o método `drawRecord()`, que exibe essa informação na parte superior direita da janela.

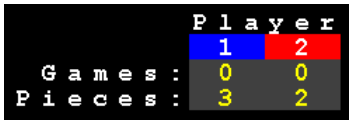


Figura 6 - Registo de vitórias e peças jogadas por cada jogador.

O método `action()` associa às teclas de comando do jogo a sua ação. As teclas *right* e *left-arrow* permitem selecionar a coluna em que se pretende colocar uma peça.

Quando a peça está no topo da coluna em que o jogador pretende jogar, este pode fazer cair a peça nessa coluna através das teclas *arrow-down* ou *space*. A peça cai até à última linha livre dessa coluna. O número de linhas que a peça cai é dado pelo total de linhas do tabuleiro menos o número de linhas preenchidas nessa coluna. Durante a queda da peça, na realidade, esta move-se uma linha de cada vez para baixo, escondendo-se na linha acima e aparecendo na linha abaixo, até estar na última linha livre (método `stepFall()` da classe `Piece`).

Uma jogada nova é efetuada quando não existem peças a cair no tabuleiro e existam posições disponíveis na coluna selecionada, isto é, quando a condição `!piece.stepFall () && availablePosition (piece.column)!=-1` for verdadeira.

Quando a peça está na posição final, essa peça passa a ser "*previous*" (preenchida com o caractere " * "), deixando a anterior peça *"previous"* de estar preenchida pelo caractere " * ". É incrementado o número de peças jogadas pelo jogador ativo em uma unidade, e é escrita a posição da peça com o número do jogador no array de registo de posições das jogadas `arrBoard[][]` (ver Listagem 2). Processa-se a troca de jogador e é gerada uma peça nova desse jogador no topo do tabuleiro, que passa a ser a peça "*last"* (preenchida com o caractere " V ").

```
public static void writeArray() {
    int avLine = availablePosition(piece.column);
    if (avLine != -1) {
        if (player)
            arrBoard[avLine][piece.column] = 2;
        else
            arrBoard[avLine][piece.column] = 1;
        printArray(arr);
    } else return;
}
```

Listagem 2 - Código do método "writeArray()".

Quando ocorre vitória de um dos jogadores ou o jogo fica empatado, é guardado numa variável o jogador que ganhou e incrementado o número de vitórias do jogador ativo no array de registo de vitórias `registoVitorias[]`. É mostrada uma mensagem com o jogador vencedor, o tipo de vitória e a opção de começar um novo jogo pressionando a tecla `ENTER`.

No caso de empate, essa informação é exibida e também é oferecida a hipótese de começar um jogo novo. Os arrays de registo de número de peças jogadas e de registo de posições das peças são então preenchidos com zeros em todas as posições, é desenhado um tabuleiro vazio e o jogo recomeça com o jogador vencedor no primeiro turno, i.e., o método `run()` é chamado recursivamente.

Para sair do jogo, o jogador pressiona a tecla `ESC` e é exibida a mensagem "GAME OVER" no centro da janela. Passados 5 segundos, ou assim que o jogador carrega numa tecla qualquer, a janela fecha e o programa termina.

Critério de vitória de um jogador

Relativamente aos critérios de vitória do jogo, foram identificadas três situações distintas: (i) um jogador faz Linha horizontal, (ii) Linha vertical ou (iii) Linha diagonal. Na figura 7 estão representadas as possibilidades de fazer Linha no caso de *N=4* (4 em Linha).

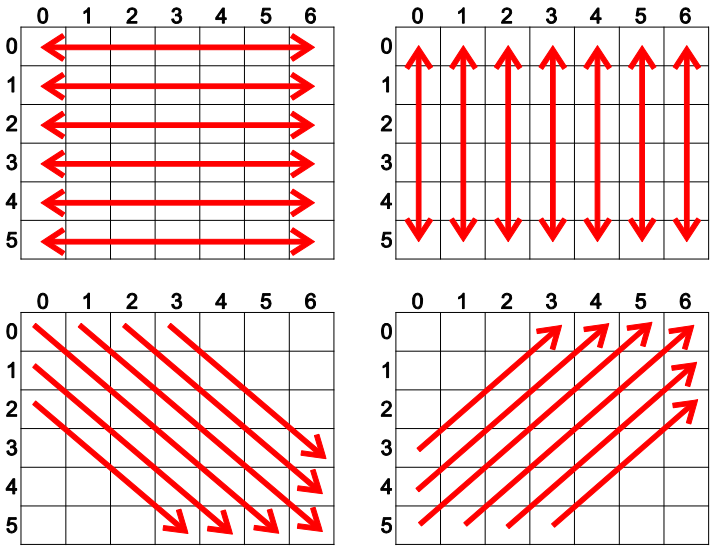


Figura 7 -Possibilidades de ocorrer Linha no tabuleiro (caso N=4).

Para a verificação de Linha horizontal, foi escrito um método `winHor()` que percorre a linha do Array (arrBoard) da última peça jogada por forma a verificar se existem "N" peças desse mesmo jogador seguidas.

De forma similar à verificação horizontal, para a Linha vertical foi escrito o método `winVert()` que percorre a coluna do Array (Board) onde a última peça foi jogada por forma a verificar se existem "N" peças desse mesmo jogador seguidas, com a particularidade de apenas percorrer entre a posição dessa mesma peça até à ultima linha do Array (i.e., do tabuleiro).

Quanto à verificação diagonal, foram identificadas duas situações distintas, i.e., a Linha diagonal poderá ser feita "descendente" ("\") ou "ascendente" ("/") (ver Figura 7). Para tal foram criados dois métodos distintos, o `winDiagAsc()` para a verificação de diagonal ascendente e o `winDiagDesc()` para a descendente. Em ambos os métodos, foram utilizados três ciclos "for" para percorrer as posições do Array onde é possível o jogador fazer Linha. Como se pode verificar na figura 7 (para o caso de N=4), estão identificadas as linhas diagonais do Array que os ciclos de "for" têm de percorrer, sendo estas dependentes do valor de N, i.e., o domínio do ciclo que percorre as linhas do Array varia, assim como o domínio do ciclo que percorre as colunas. Na tabela seguinte está descrito o domínio dos ciclos presentes nos métodos.

Tabela 1 - Tabela do domínio dos ciclos de "for" para a verificação de Linha na diagonal (métodos winDiagDesc() e winDiagAsc()).

	winDiagDesc()				**winDiagAsc()**			
N	**Linha inicial**	**Linha Final**	**Coluna inicial**	**Coluna final**	**Linha inicial**	**Linha Final**	**Coluna inicial**	**Coluna final**
3	0	2	0	2	4	2	0	2
4	0	2	0	3	5	3	0	3
5	0	3	0	4	7	4	0	4

É de salientar que no caso N=3, tabuleiro do jogo toma um valor de linhas igual à de colunas.

Foi ainda escrito um terceiro ciclo que percorre mais N-1 posições na diagonal do Array quando é encontrada uma peça do jogador que jogou, contendo este uma variável contadora que verifica se existem N peças iguais consecutivas. Na figura seguinte é feita uma representação gráfica de um exemplo do funcionamento do método `winDiagDesc()`.

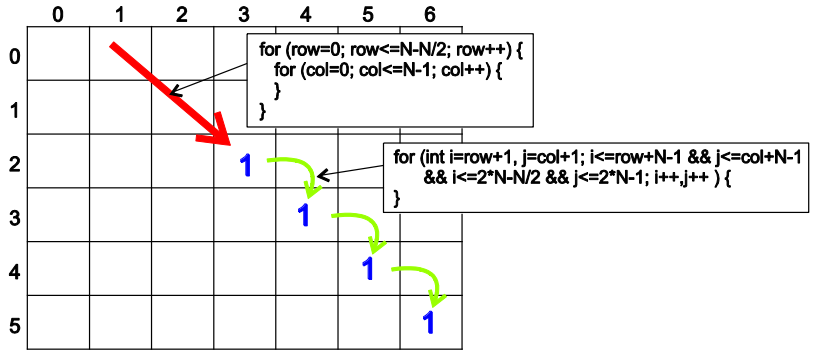


Figura 8 - Representação gráfica dos ciclos "for" implementados no método winDiagDesc().

Posto o acima explanado, e a titulo de exemplo, na listagem 3 é mostrado o código do método `winDiagAsc()`.

```
   private static boolean winDiagAsc(int player) {
    int row, col;
    for (row=2*N-N/2-1; row>=N-1; row--) {
        for (col=0; col<=N-1; col++) {
            if (arr[row][col] == player) {
                int contadorWin = 1;
                for (int i=row-1, j=col+1; i>=row-N-1 && j<=col+N-1 &&
i>=0 && j<=2*N-1; i--,j++ ) {
                    if (arr[i][j] == player) contadorWin += 1;
                    if (contadorWin >= N) return true;
                }
            }
        }
    }
    return false;
}
```

Listagem 3 - Código do método winDiagAsc().

Conclusões

Com a realização deste trabalho, foi possível entender as diferenças entre os paradigmas de programação estruturada, até então usada para a realização dos trabalhos anteriores, e de programação orientada a objectos. A programação orientada a objectos organiza o programa de uma forma similar ao mundo real, em que os objectos estão associados a atributos e a actividades, i.e., ao contrário da programação estruturada que coloca os dados e as operações separados, a programação orientada a objectos coloca as operações e os dados a que pertencem num objecto. Este paradigma torna o programa mais flexível, modular, com código reutilizável, mais fácil de manter (actualizações), mais fácil de desenvolver permitindo a um programador desenvolver parte do código sem conhecer todo o código do programa.

Com este trabalho foram aplicados os conceitos de classes em Java. Uma classe é um conjunto de dados e métodos que operam sobre esses mesmos dados. Um objecto é uma instância da classe, ou seja, podem criar-se várias instancias da classe, i.e. vários objectos.

Durante o desenvolvimento da aplicação foram identificados dois erros que foram corrigidos:

- Na eventualidade de um jogador querer sair do jogo, identificou-se um erro do tipo `NullPointerException` no método `action`. Tal acontecia, pois o método `terminate` espera 5 segundos para que seja pressionada qualquer tecla, ao final dos quais fecha a janela do jogo. Foi adicionado o comando `System.exit(0)` depois do método `terminate` para corrigir esta situação.
- Na primeira jogada de um jogo que não fosse o primeiro, durante a queda da primeira peça, surgia no tabuleiro a última peça do jogo anterior. Este erro foi corrigido identificando as condições irrepetíveis que se davam nessas circunstâncias. Como apenas na primeira jogada as duas posições do array de registo de numero de peças jogadas estão preenchidas a zero, e num jogo que não o primeiro, a variável `player` toma o mesmo valor que `winningplayer`, adicionaram-se essas condições ao método `fix` da classe `Piece`, aplicando somente o método `show` à peça `last`. Assim, impediu-se que o método `show` fosse aplicado à peça `previous` na primeira jogada.

Ainda de salientar, quando o jogador reúne as condições de vitória na diagonal, foi necessário dividir o problema em duas partes distintas: (i) quando a Linha diagonal é ascendente ou (ii) descendente. Na solução deste ponto foram escritos dois métodos distintos (`winDiagAsc()` e `winDiagDesc()`).

A programação não é independente do jogo propriamente dito, ou seja, foi necessária uma compreensão abrangente do jogo por forma a identificar todas as particularidades do mesmo (critérios de vitória etc.).

Em suma, este trabalho mostrou-se um bom exercício de aplicação e entendimento da estrutura de um programa utilizando o paradigma da programação orientada a objectos, e a aplicação de uma diferente forma de interação com o utilizador usando a classe ConsolePG para leitura de dados introduzidos e escrita (*input-output*).

Anexo – Funcionalidades Opcionais

Foram propostas duas funcionalidades opcionais: (i) a possibilidade de o jogador **anular** a última jogada efetuada; e (ii) jogar contra o computador, em 4 níveis **automáticos**, de crescente dificuldade.

Para anular a última jogada foi necessário recorrer a uma variável booleana `undone`, que sinaliza se já foi desfeita uma jogada no turno presente ou não, e um array de objetos `lastthree`, que guarda as últimas três jogadas efetuadas no jogo.

Esta funcionalidade está distribuida, do ponto de vista da conceção e estrutura do programa, por três pontos: o método `fix` e o método `undoLast`, ambos da classe `Piece`, e o método `action` da classe `InLine`.

No método `fix`, enquanto não for desfeita jogada, (i.e, `undone == false`) o array de registo das últimas 3 jogadas é atualizado. Para tal, cada peça é colocada no indíce +1 do array. O objeto no índice 0 passa sempre a ser a última jogada. A variável `undone` toma o valor "false".

Foi acrescentado um caso ao método `action`. Assim, sempre que for acionada a tecla seta para cima (`key==KeyEvent.VK_UP`), se não foi já desfeita uma jogada nesse turno, o método `undoLast` é chamado. Se não se estiver num dos níveis automáticos, o jogador ativo é trocado. É criada uma nova peça no topo do tabuleiro.

No método `undoLast`, é escondida a última jogada, é apagado o registo dessa jogada no array de registo do estado do tabuleiro, decrementado o número de peças jogadas do jogador cuja jogada foi desfeita. Se se estiver num dos níveis automáticos, é repetido o processo para a penúltima jogada, de modo a anular a jogada feita pelo utilizador, bem como a jogada automática que lhe sucedeu. Ainda nesse caso, a peça `last` passa a ser a antepenúltima. Se se estiver no modo dois jogadores, a peça `last` passa a ser a penúltima. É mostrada a nova última peça jogada (preenchida com *), `undone` passa a tomar o valor `true`.

```
   public void undoLast(){

       if (  undone == false){ //the move wasn't undone
           lastthree[0].hide();      //hides last piece played
           if (InLine.autolevelcounter!=0){
               lastthree[1].hide();//hides second to last piece played
(only in auto mode)
InLine.arrBoard[lastthree[1].getLine()][lastthree[1].getColumn()] =0;
               --InLine.registoPieces[InLine.playernum(InLine.player)-
1];
               last= lastthree[2];
           }

           else last = lastthree[1];

InLine.arrBoard[lastthree[0].getLine()][lastthree[0].getColumn()] =0;
           --InLine.registoPieces[InLine.playernum(!InLine.player)-1];
           last.show();
           undone = true;
           this.hide();
       }
       return;
   }
```

Método undoLast da classe Piece 1

Como se pretendeu aproveitar a modularidade do trabalho, reaproveitando parte do código utilizado anteriormente para implementar esta funcionalidade, nomeadamente os métodos que determinam se houve ou não vitória, conceberam-se quatro níveis de jogo automático que se distinguem pelas seguintes características:

1) É gerada uma variável aleatória entre 0 e o número de colunas do tabuleiro de jogo. A jogada é feita na coluna correspondente. (Note-se que se considera que a coluna de índice zero é a primeira coluna);
2) É procurada uma peça própria. Se houver possibilidade de vitória nessa coluna, é jogada aí uma peça. Se não, verifica-se se há possibilidade de vitória na mesma linha, para aí jogar. Em caso negativo, é accionado o nível automático um;
3) É procurada uma ameaça, i.e, uma sequência de N-1 ou N-2 peças na vertical, horizontal, ou diagonal. É feita uma jogada de modo a neutralizar essa ameaça. Se tal não suceder, é accionado o nível automático dois;
4) É seguida a estratégia ganhadora descrita por Allis 1988, tendo em conta o controlo de Zugzwang, um evento no jogo em que o jogador ativo tem duas ameaças de N-1 peças para defender, ganhando o outro jogador na ronda seguinte, e regras estratégicas descritas pelo mesmo autor.

Com a disponibilidade havida, foram elaborados os dois primeiros níveis automáticos. Havendo mais disponibilidade de tempo, propunha-se que se avançasse para completar os dois restantes níveis.

Referências

[1] Savitch W. *Java: An Introduction to Problem Solving and Programming*. 7th Ed ed: Pearson; 2015.

[2] Java Documentation - Java Platform, Standard Edition (Java SE) 8. https://docs.oracle.com/javase/8/. Accessed 29-10-2016.

[3] Liang, Y. Daniel. Introduction to Java™ Programming, Comprehensive Version. Ch. 8, 9,10. 9th Ed ed:Pearson; 2013.

[4] Victor Allis. A Knowledge-based Approach of Connect-Four The Game is Solved: White Wins. Department of Mathematics and Computer Science Vrije Universiteit Amsterdam, The Netherlands Masters Thesis, October 1988.