

**Engenharia Informática**

**Programação Orientada a objetos**

**2019/2020**

**Aplicação BlockuDocku**

**Realizado por**: Eduardo Ferreira nº110221031

Duarte Conceição nº

**Docente:** Noémia Ferro

Índice

[Introducão 3](#_Toc41390494)

[Descrição da Aplicação 4](#_Toc41390495)

[Estrutura do programa 5](#_Toc41390496)

[Explicação de métodos importantes 7](#_Toc41390497)

[Conclusão 8](#_Toc41390498)

# **Introducão**

Neste relatório técnico irá ser fornecido uma breve descrição de todas as funcionalidades da aplicação/jogo Blockudoku, assim como explicações bem detalhadas em relação á sua estrutura, classes utilizadas e certos métodos que possuem uma importância mais elevada para o funcionamento do projeto.

# **Descrição da Aplicação**

A aplicação Blockudoku é um jogo de tabuleiro altamente inspirado em dois jogos extremamente conhecidos e populares (Sudoku e Tetris), e de certa forma é uma mistura de ambos possuindo o que torna cada um especial e no entanto mantem/possui a sua própria identidade.

Assim como no tetris, este jogo é jogado através de peças geométricas que podem ser rodadas em incrementos de 90º (90º, 180º, 270º, 360º), no entanto ao contrário do tetris, o jogador não possui controlo nenhum sobre a rotação das peças, em vez disso será o próprio jogo que deverá rodar as peças autonomamente e automaticamente antes de as entregar ao jogador para as colocar no tabuleiro. No entanto como o tetris a pontuação será atribuída sempre que o utilizador consiga fazer/criar uma linha ou coluna completa no tabuleiro, eliminando essa linha e atribuindo pontos por isso. A colocação das peças é direta, isto significa que ao contrario do tetris onde as peças caiem verticalmente para o tabuleiro, este jogo simplesmente coloca-as nas coordenadas escolhidas caso seja possível.

Em relação ao sudoku e as semelhanças com este jogo, embora Blockudoku não possuía números mas em vez disso peças para colocar no tabuleiro, é o tabuleiro em si que está relacionado com o jogo, o tabuleiro consiste em 81 casas (9x9) que está divido em 9 (áreas 3x3) tendo assim a mesma semelhança de “mini área” que o sudoku possui, e quando uma dessas mini áreas é completamente preenchida o jogador receberá pontos extra.

# **Estrutura do programa**

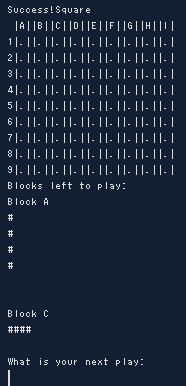
O grupo esforçou-se a tentar possuir uma aplicação/código altamente coeso e pouco acoplado, tendo cada classe apenas o código necessário para funcionar e os seus atributos/métodos respectivos tornado assim mais fácil o desenvolvimento e correção de erros sem ter a necessidade de alterar todas as outras classes.

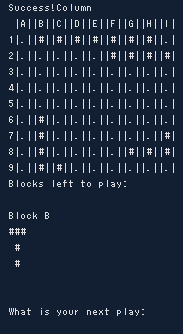
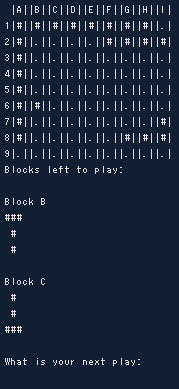
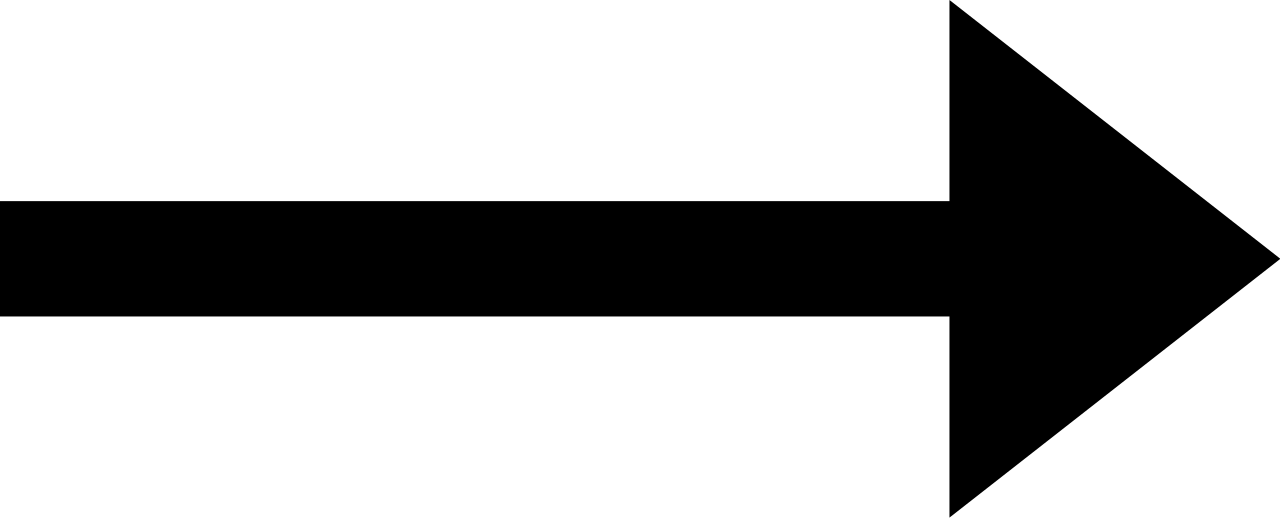
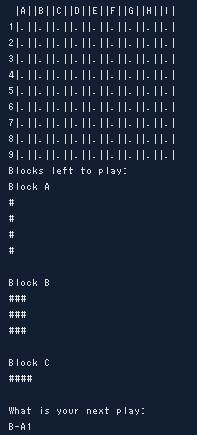
Inicialmente o grupo tinha optado por criar uma interface Block que iria possuir todos os métodos abstratos, fazendo com que as classes que implementam certa interface deveriam possuir esses métodos, isso juntamento com superclasses que alguns blocos iriam ser (por exemplo blocos como o T e o TExtended) de forma a possuir uma estrutura que utiliza tanto interfaces como superclasses para mostrar uma boa utilização de herança. No entanto o grupo decidiu abandonar a utilização de interface quando se notou que era possível reduzir consideravelmente o código utilizado se optássemos por tornar a interface numa superclasse, possuindo assim uma hierarquia de classes onde a Classe Block possui maioria do código e as superclasses que anteriormente implementavam a interface (BlockT, BlockQ. BlockI, etc) teriam maioria do código removido.

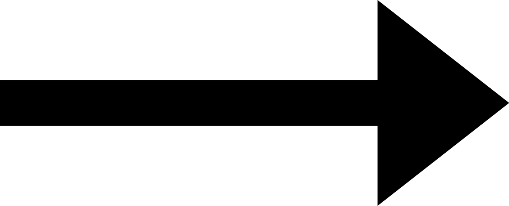
Em relação á rotação o grupo optou por utilizar uma class enumerada, que iria ter os 3 tipos de rotações pedidos no guia do projeto, esta classe possui um método que devolve o número de rotações de 90º que a peça deverá fazer simulando assim rotações de 90/180/270 utilizando o método values() dos enumerados que devolve um array com todos os valores do enumerado sendo assim possível escolher um índex aleatório.

A classe Round (Rodada) será a classe que trata de preparar e fornecer ao jogo os 3 blocks que iram ser utilizados na rodada atual, existem alguns métodos que irão ser explicados com mais detalhe no tópico a seguir, mas basicamente esta classe obtém 3 blocks aleatórios prepara-os.

A classe BlockuDoku (classe jogo) é o nosso main que basicamente faz com que as outras classes estejam a correr sem problemas, esta classe possui ligações diretas com o player e com a classe Round, o grupo optou por esta logica pois pensamos que fazia mais sentido ser o “jogo” que manda preparar a rodada e após recebida atribui ao jogador em vez de ter a classe round diretamente ligado ao jogador.

A classe player possui apenas os métodos/comportamentos que o jogador pode fazer, que simplesmente se trata de enviar uma peça escolhida para a classe de tabuleiro (board) assim como puder realizar as operações de save /load/watch personal scores/ ranking. Por fim existe também a classe tabuleiro que trata de receber a peça vinda do jogador e verifica se é possível coloca-la, para alem disso faz também as verificações necessárias para saber se uma linha / coluna/ Big square foi pre-enchido.





# **Explicação de métodos importantes**

A classe Round (rodada) possui diversos métodos importantes para a criação de um bloco que tenha sofrido uma rotação. Como a rotação dos blocos foi baseada em resolução de matrizes (nomeadamente rodar matrizes através de um pivot x, y) foi necessário criar os blocos num array bidimensional com linhas e colunas extra para que a rotação possa ser efectuada, devido a isso foi então criado um conjunto de métodos especiais focados em “limpar” o array bidimensional o máximo possível apos a rotação. Primeiramente possuímos os métodos moveUp() e moveLeft(), estes métodos deveram mover o bloco no array o máximo possível para o topo e esquerda respectivamente fazendo assim com que exista linhas vazias em baixo e colunas vazios á direita. Para remover essas linhas em excesso foi então criado os métodos removeRows e removeColumns que deverão remover todas as linhas e colunas vazias, fazendo com que o bloco que seja entregue ao jogador tenho o mínimo de tamanho possível, facilitando assim a sua colocação no tabuleiro.

A class Board também possui alguns métodos importantes que vale a pena mencionar, nomeadamente os métodos verifyScore(), que invoca três métodos específicos para verificar se uma linha/coluna/Quadrado grande, foi preenchido, esses métodos são respetivamente: verifyLine(int line), verifyColumn(int line), verifyBigSquare(int line, int column). O grupo optou por separar a verificação de espaços preenchidos de forma a dividir e simplificar o problema que seria verificar todas as condições ao mesmo tempo. Como é possível que exista combinações de preenchimentos (um Quadrado grande e uma linha ou coluna ao mesmo tempo), os métodos de verificação embora sejam realizados sequencialmente as remoções só serão realizados apos todas as verificações para caso algo deste genro suceda onde seja necessário atribuir pontos por eliminação de preenchimentos completos.

Por fim na classe player também existe alguns métodos que vale a pena explicar/anunciar, nomeadamente o método ranking() que utiliza métodos específicos das coleções (sort e reverse) para ordernar uma lista, primeiramente por ordem ascendente e de seguida de forma decrescente de forma a obter os maiores valores de cada jogador.

# **Conclusão**

Em suma, embora o projeto tenha tido os seus desafios eu penso que o grupo conseguiu resolver/implementar tudo o que era necessário/pedido para que o programa pudesse funcionar sem quaisquer problemas. Penso que a utilização de heranças nos blocks foi adequada pois reduziu bastantemente o código que seria necessário caso não tivesse sido feito, a criação de listas para puder aceder ás collections também facilitou a realização do projeto bastante.