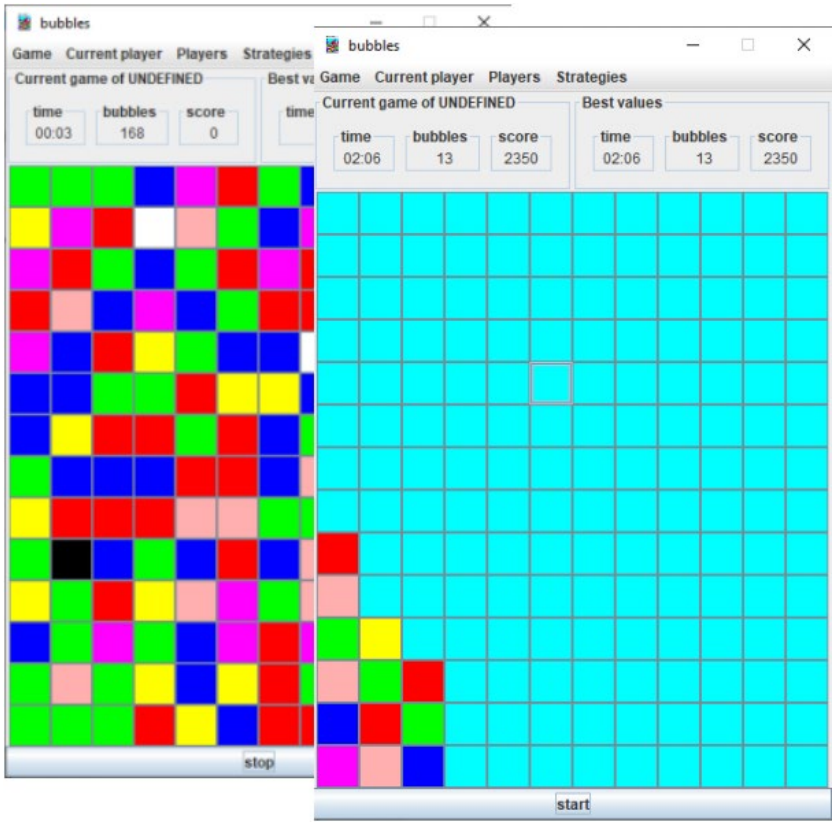
Programação III

Licenciatura em Engenharia Eletrónica e Telecomunicações e de Computadores

3º Trabalho Prático

Jogo Bubbles



Grupo 22 LT41D

45866 Daniel Barradas

45880 Bernardo Pineda

2º Semestre letivo 2020/2021

10 de julho de 2021

Índice

1. Descrição

Componentes Principais do Jogo

i. BubbleGameFrame

A classe BubbleGameFrame diz respeito a toda a parte gráfica do jogo. Desde a apresentação dos menus e dos botões até a visualização do tabuleiro e das mudanças nos tabuleiros após eventos.

ii. BubbleGame

A classe BubbleGame diz respeito a toda a lógica do jogo. Nesta classe é definido o tabuleiro, a percentagem de cada cor, a posição onde serão colocadas as bolhas e o tipo de estratégia a ser implementado. É ainda nesta classe que é feito a atualização do número de bolhas e da pontuação. Esta classe dispõe de um conjunto de listeners para que seja capaz de aperceber de mudanças no jogo, especialmente, do score e do tempo de jogo.

iii. Strategty

A Strategy é uma interface que deve ser implementada por classes que definam um tipo de estratégia. Por exemplo a classe StrategyGravitational, que implementa a interface Strategy, define um tipo de estratégia de jogo. Independentemente do tipo de estratégia, gravitational, shifter ou outra, qualquer classe que defina uma estratégia tem a mesma necessidade de implementar métodos como select() e removeSelected() que definem a forma como devem ser removidas as bolhas.

Interação

Como foi referido acima, a classe BubbleGameFrame diz respeito a toda a parte visual do jogo, assim como eventos como carregar em botões. Quando é executado o BubbleGameFrame é passado ao construtor um jogo e uma estratégia, que são respetivamente, um BubbleGame e uma Strategy. Quando se efetua uma ação como selecionar uma bolha, a parte visual, isto é, por exemplo, a bolhas ficarem cinzentas, feita no BubbleGameFrame, no entanto, toda a lógica de jogo, como saber que bolhas deve selecionar quando se carrega num tipo de bolha, é feita através da classe BubbleGame que por sua vez vai depender do tipo de estratégia do jogo, por exemplo, para saber para que posição se devem mover as peças após a remoção das selecionadas.

Listeners

i. BubbleListener

O BubbleListener é uma interface que declara 2 métodos, select() e unselect(). Este é o Listener mais utilizado no jogo, uma vez que diz respeito à própria essência do jogo, isto é, de selecionar bolhas. Esta interface é estendida por 2 outras interfaces, o GameListener e o Board.

O GameListener diz respeito a mudanças na pontuação ou no tempo (começo e termino do jogo), enquanto o Board diz respeito a mudanças no tabuleiro, isto é, colocação de bolhas ou de buracos e ainda a localização da posição das bolhas. Esta hierarquia é uma boa solução uma vez que ambos os listeners dizem respeito a mudanças nas bolhas (BubbleListener) no entanto um deve atuar sobre a pontuação quando estas mudanças acontecem (GameListener) e o outro deve atuar sobre o tabuleiro (Board).

O Board é implementado pela classe BubbleGame uma vez que este faz toda a lógica do jogo e deslocamento de bolhas no tabuleiro.

O GameListener é implementado no BubbleGameFrame, uma vez que este deve fazer a apresentação da pontuação.

Repare no entanto que em ambos os casos, terão de ser implemtandos os métodos select() e unselect(), pois, como explicado anteriormente, tanto o GameListener como o Board, estendem o BubbleListener;

ii. GameListener

O Game Listener é uma interface que diz respeito a mudanças como começar um jogo, acabar um jogo e mudanças na pontuação e, por isso, declara 3 métodos, scoreChange(), gameStop() e gameStart(). Uma vez que o GameListener estende a Interface BubbleListener, todas as classes que implemente o GameListener terão de definir os métodos da interface BubbleListener, para além de definirem os métodos do GameListener.

2. Outros Componentes do Jogo

Player

i. Hierarquia

A classe BubbleGameFrame tem um mapa de Player (players: Map<String, Player>) e um Player (currPlayer).

ii. Membros

Cada Player tem um nome (name: String), estatísticas dos seus jogos (statistics: Statistics) e a melhor pontuação (bestScore: Score).

iii. Métodos

O Player tem dois construtores, que inicializam os seus membros. Existem dois construtores, um que recebe o nome do jogador e outro para o caso de não ser especificado o nome do jogador (o nome fica "UNDEFINED");

O Player tem métodos:

- para aceder aos seus membros, getters, getName(), getBestScore(), getStatistics();
- para definir a melhor pontuação do jogador, setBestScore();
- que permite imprimir a informação de vários jogadores armazenados num mapa, printPlayers();
- que retorna uma string com informação sobre os jogadores com as 10 melhores pontuações, getTopX();

Statistics

i. Hierarquia

Cada jogador (Player), tem as estatísticas dos seus jogos (statistics) que são uma instância de Statistics.

ii. Membros

A estatísticas consistem num conjunto de valores inteiros:

- tGames – número de total de jogos jogados;

- wGames – número de jogos ganhos;
- maxScore – pontuação máxima obtida;
- avgScore – media das pontuações obtidas;
- minTime – tempo mínimo obtido;
- avgTime – média dos tempos obtidos;
- maxBubbles – número máximo de bolhas não destruídas;
- minBubbles – número mínimo de bolhas não destruídas;
- avgBubbles – media do número de bolhas não destruídas;

iii. Métodos

Quando é criada uma instância de Statistics, todos os membros descritos acima são inicializados no construtor.

Esta classe dispõe do método addScore(), que permite fazer a atualização dos membros caso seja necessário, assim como calcular o novo valor média dos membros, através do método calcAvg(). Dispoe ainda de um método clear, que reinicializa os valores dos membros e de um método toString() que retorna uma String formatada com as apresentação dos valores dos membros.

ValuesPanel

i. Hierarquia

Na BubbleGameFrame existem dois painéis (currPanel e bestPanel) que são instacias de ValusPanel. Estas variáveis servem para a apresentação, na Frame da aplicação, do score do jogo atual assim como a do melhor score obtido pelo jogador.

ii. Membros

O painel de valores é constituído pelos valores do score (tempo, número de bubbles e pontuação):

- tfTime – campo de texto para o tempo do jogo (é um timer em tempo real no currPanel e é fixo no bestPanel);
- tfBubbles – campo de texto para número de bolhas restantes (é atualizado ao longo do jogo no currPanel e é fixo no bestPanel);
- tfScore – campo de texto para a pontuação (é atualizado ao longo do jogo no currPanel e é fixo no bestPanel);

iii. Métodos

Quando é criada uma instância de ValuesPanel, todos os membros descritos acima são inicializados no construtor.

Esta classe tem apenas getters que servem para aceder aos membros das instacias da classe, getTimer(), getBubbles() e getScore().

NeighborBubble

iv. Hierarquia

Esta classe foi criada devido à semelhança que havia entre as classes CrossBubble e DiagonalBubble. Ambas as classes mencionadas têm características semelhantes e por isto necessitavam de os mesmos métodos diferindo apenas nas suas direções. Esta classe estende a classe SelectableBubble.

v. Membros

Cada NeighborBubble tem uma cor (color).

vi. Métodos

A cor da bubble é inicializada quando é criada uma instância desta classe.

Esta classe dispõe de dois getters, getColor() e getDirections(). Note que o segundo getter referido é abstrato, uma vez que, as direções diferem dependendo do tipo de NeighborBubble, - CrossBubble ou DiagonalBubble - como mencionado acima, devendo ser definido nestas classes.

Esta classe dispõe ainda de métodos que servem para a seleção das bolhas, select(), usado no selectIf() e o selectAround().

Note que, como mencionado acima, tanto o CrossBubble como o DiagonalBubble foram modificados. Ambas estas classes passaram a estender a classe criada, NeighborBubble.

Ambas as classes apenas dispõem do construtor e de um método getter que retorna as direções, getDirections().

3. Estruturas de Dados das Estatísticas

Dados

Cada jogador necessita das suas estatísticas de jogo, podendo existir vários jogadores.
Então foi criado um mapa de jogadores (Map<String, Player> players) no BubbleGameFrame. Foi escolhido um mapa, devido à rapidez de acesso aos elementos do mesmo, O(1), sendo então o acesso aos jogadores mais rápido e eficiente.
Cada jogador tem um nome, estatísticas e a sua melhor pontuação. A classe Player dispõe de métodos que acedem ou modificam estes membros. Existem ainda outros métodos que atuam sobre as estatísticas de cada Player.
Para as estatísticas de cada jogador, foi desenvolvida uma classe, Statistics. Cada jogador tem um membro (statistics) que é uma instância desta classe.
A classe Statistics dispõe de vários membros que dizem respeito às estatísticas do jogo. Os métodos desta classe são métodos que atuam sobre esses membros, adcionando novos valores, addScore() ou reinicializando esses valores, clear().

Métodos mais relevantes

Statistics:

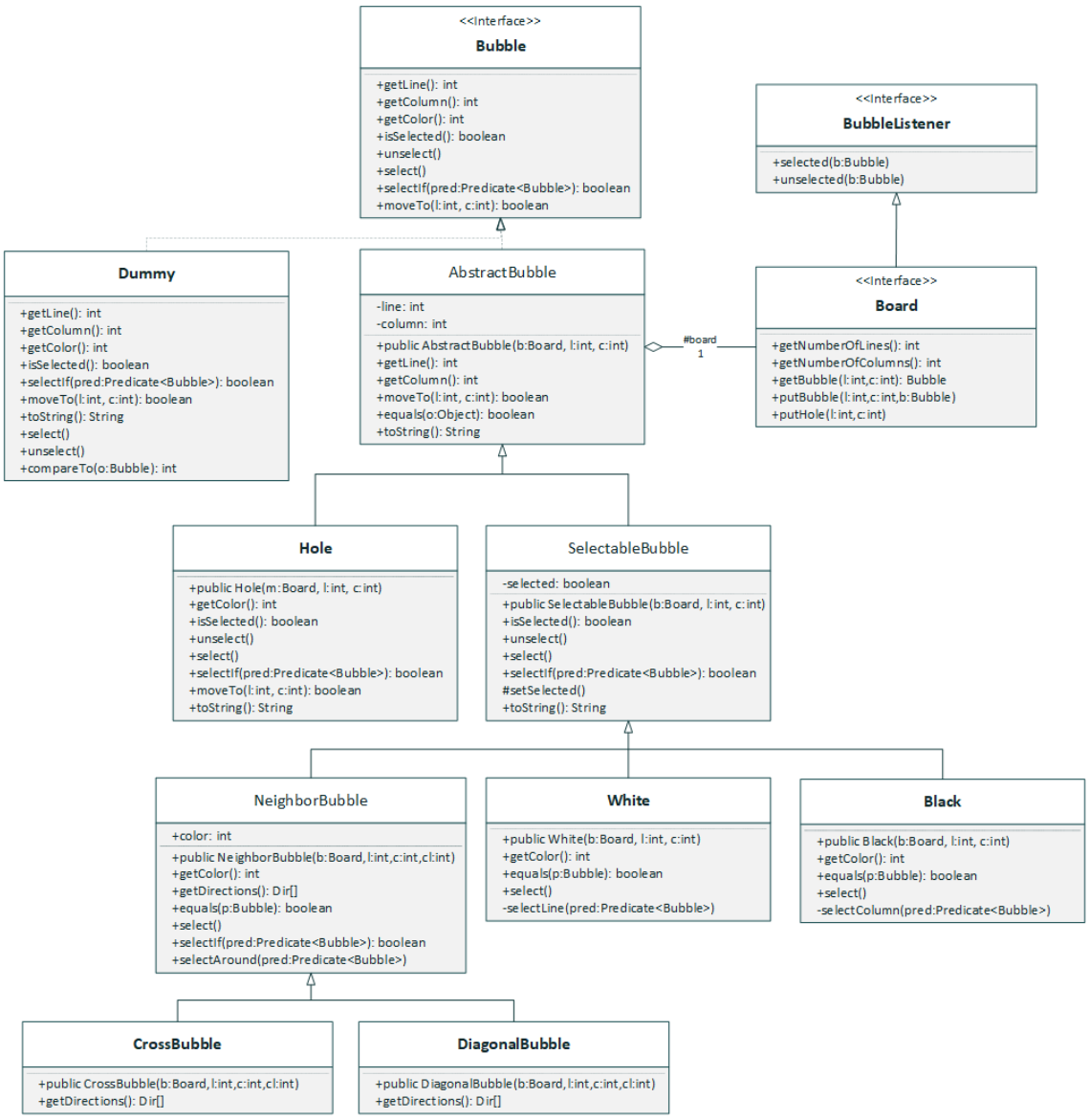
- addScore(), clear() e saveStats();

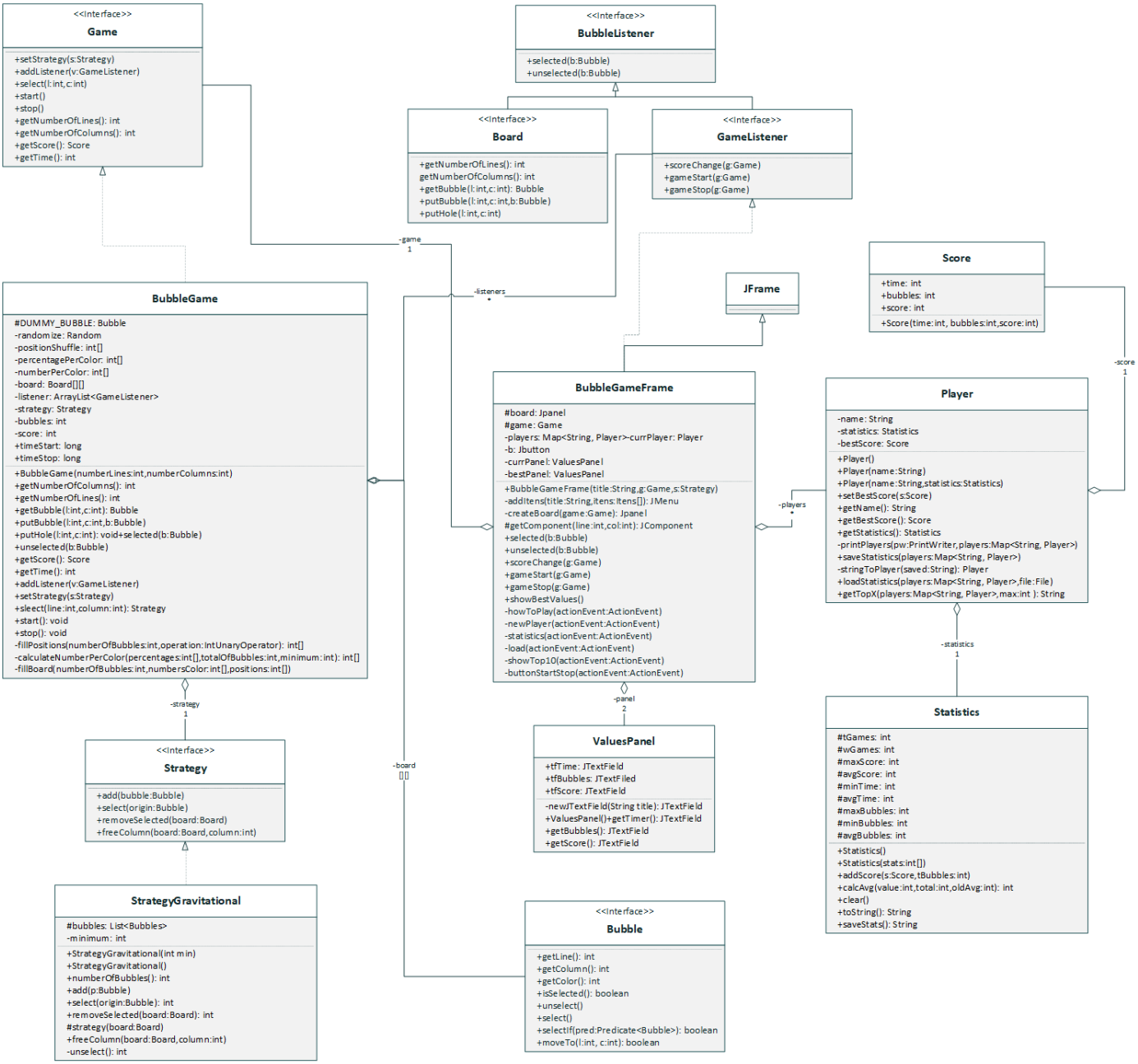
Player:

- setBestScore(), printPlayers(), getTopX() e loadStatistics();

Os membros e os métodos de ambas as classes, Player e Statistics, foram descritos em maior detalhe no ponto 2. Outros Componentes do Jogo, deste relatório.

4. Diagramas UML




<<Interface>>
Game
+setStrategy(s:Strategy)
+addListener(v:GameListener)
+select(l:int,c:int)
+start()
+stop()
+getNumberOfLines(): int
+getNumberOfColumns(): int
+getScore(): Score
+getTime(): int
<<Interface>>
BubbleListener
+selected(b:Bubble)
+unselected(b:Bubble)
<<Interface>>
Board
+getNumberOfLines(): int
getNumberOfColumns(): int
+getBubble(l:int,c:int): Bubble
+putBubble(l:int,c:int,b:Bubble)
+putHole(l:int,c:int)
<<Interface>>
GameListener
+scoreChange(g:Game)
+gameStart(g:Game)
+gameStop(g:Game)
-game
1
-listeners
*
Score
+time: int
+bubbles: int
+score: int
+Score(time:int, bubbles:int,score:int)
JFrame
BubbleGame
#DUMMY_BUBBLE: Bubble
-randomize: Random
-positionShuffle: int[]
-percentagePerColor: int[]
-numberPerColor: int[]
-board: Board[][]
-listener: ArrayList<GameListener>
-strategy: Strategy
-bubbles: int
-score: int
+timeStart: long
+timeStop: long
+BubbleGame(numberLines:int,numberColumns:int)
+getNumberOfColumns(): int
+getNumberOfLines(): int
+getBubble(l:int,c:int): Bubble
+putBubble(l:int,c:int,b:Bubble)
+putHole(l:int,c:int): void+selected(b:Bubble)
+unselected(b:Bubble)
+getScore(): Score
+getTime(): int
+addListener(v:GameListener)
+setStrategy(s:Strategy)
+sleect(line:int,column:int): Strategy
+start(): void
+stop(): void
-fillPositions(numberOfBubbles:int,operation:IntUnaryOperator): int[]
-calculateNumberPerColor(percentages:int[],totalOfBubbles:int,minimum:int): int[]
-fillBoard(numberOfBubbles:int,numbersColor:int[],positions:int[])
BubbleGameFrame
#board: Jpanel
#game: Game
-players: Map<String, Player>-currPlayer: Player
-b: Jbutton
-currPanel: ValuesPanel
-bestPanel: ValuesPanel
+BubbleGameFrame(title:String,g:Game,s:Strategy)
-addItens(title:String,itens:Itens[]): JMenu
-createBoard(game:Game): Jpanel
#getComponent(line:int,col:int): JComponent
+selected(b:Bubble)
+unselected(b:Bubble)
+scoreChange(g:Game)
+gameStart(g:Game)
+gameStop(g:Game)
+showBestValues()
-howToPlay(actionEvent:ActionEvent)
-newPlayer(actionEvent:ActionEvent)
-statistics(actionEvent:ActionEvent)
-load(actionEvent:ActionEvent)
-showTop10(actionEvent:ActionEvent)
-buttonStartStop(actionEvent:ActionEvent)
-players
*
-score
1
Player
-name: String
-statistics: Statistics
-bestScore: Score
+Player()
+Player(name:String)
+Player(name:String,statistics:Statistics)
+setBestScore(s:Score)
+getName(): String
+getBestScore(): Score
+getStatistics(): Statistics
-printPlayers(pw:PrintWriter,players:Map<String, Player>)
+saveStatistics(players:Map<String, Player>)
-stringToPlayer(saved:String): Player
+loadStatistics(players:Map<String, Player>,file:File)
+getTopX(players:Map<String, Player>,max:int): String
-statistics
1
-strategy
1
-panel
2
-board
[][]
<<Interface>>
Strategy
+add(bubble:Bubble)
+select(origin:Bubble)
+removeSelected(board:Board)
+freeColumn(board:Board,column:int)
ValuesPanel
+tfTime: JTextField
+tfBubbles: JTextFiled
+tfScore: JTextField
-newJTextField(String title): JTextField
+ValuesPanel()+getTimer(): JTextField
+getBubbles(): JTextField
+getScore(): JTextField
Statistics
#tGames: int
#wGames: int
#maxScore: int
#avgScore: int
#minTime: int
#avgTime: int
#maxBubbles: int
#minBubbles: int
#avgBubbles: int
+Statistics()
+Statistics(stats:int[])
+addScore(s:Score,tBubbles:int)
+calcAvg(value:int,total:int,oldAvg:int): int
+clear()
+toString(): String
+saveStats(): String
StrategyGravitational
#bubbles: List<Bubbles>
-minimum: int
+StrategyGravitational(int min)
+StrategyGravitational()
+numberOfBubbles(): int
+add(p:Bubble)
+select(origin:Bubble): int
+removeSelected(board:Board): int
#strategy(board:Board)
+freeColumn(board:Board,column:int)
-unselect(): int
<<Interface>>
Bubble
+getLine(): int
+getColumn(): int
+getColor(): int
+isSelected(): boolean
+unselect()
+select()
+selectIf(pred:Predicate<Bubble>): boolean
+moveTo(l:int, c:int): boolean

5. Temas Estudados

Herança e Polimorfismo

É possível ver herança em quase todos os módulos deste trabalho. Por exemplo, a classe AbstractBubble é estendida por duas classes Hole e SelectableBubble. Isto é útil uma vez que tanto um Hole como um SelectableBubble têm posições que dependem de linhas e colunas, ou seja, tem a mesma forma de identificar a sua posição, então é definida uma classe que que tenha métodos para esse efeito, classe essa que é "mãe" de ambas as classes Hole e SelectableBubble.
É ainda possível observar que a classe SelectableBubble é estendida por três classes, NeighborBubble, White e Black, que são três tipos de bolhas que podem existir num jogo de bubbles. Este foi um exemplo de herança, no entanto, é possível observar outros exemplos ao longo do trabalho.

O polimorfismo acontece quando uma classe declara um método abstrato. Como referenciado anteriormente, as classes filhas, terão acesso aos métodos da classe mãe. No entanto quando se trata de um método abstrato, este deve ser obrigatoriamente definido pelas classes que estendem a classe mãe (as filhas). A vantagem é que para cada classe filha esse método pode ser definido de forma diferente, dependendo da sua aplicação. Isto pode ser observado Na classe NeighborBubble, que tem o método abstrato getDirections(), e que é estendida pela classe CrossBubble e DiagonalBubble. Em ambas as classes filhas, o método getDirections() é definido, no entanto de formas diferentes.

Estruturas de Dados

Neste trabalho foram usados maioritariamente dois tipos de estruturas de dados.

- Map (HashMap) – este tipo de estruturas implementa a interface Cloneable e Serializable. Neste tipo de estruturas existe uma chave e um valor para cada chave. É uma estrutura dinâmica onde o acesso é rápido. Podemos ver uma utilização por exemplo no mapa de jogadores (Map<String, Player> players). Foi escolhido ste tipo para os jogadores uma vez que o mais importante era o acesso;
- List (ArrayList) - este tipo de estruturas implementa a interface Cloneable, Serializable, Iterable e Collection. Neste tipo de estrutura a inserção e remoção de elementos é rápida e eficiente. Podemos ver um exemplo da utilização na lista de bubbles (List<Bubble> bubbles). Foi escolhido este tipo para as bubbles uma vez que o mais importante era a inserção e remoção de bolhas.

Interfaces Funcionais

Uma interface permite definir métodos que tem de ser implementados nas classes que implementem a interface. Isto é útil, uma vez que é definido uma estrutura a seguir, e apesar de em diferentes classes os métodos poderem ser definidos de formas diferentes, as suas utilizações serão idênticas.

Podemos ver um exemplo de uso de interfaces ao longo do programa. Existem as interfaces Bubble, Board, Game, Strategy, GameListener e BubbleListener.

A interface Bubble é implementada pela classe AbstractBubble.

A interface Strategy é implementada por duas classes diferentes, e a implementação dos métodos é diferente apesar de ambos receberem os mesmos paramentos e retornarem o mesmo tipo de dados.

Streams

Streams são objetos. Os dados escritos num output stream são entregues num destino que pode ser um ficheiro, o standard output (ecrã), etc. Os dados que o programa lê de um input stream são recebidos duma fonte que pode ser um ficheiro, o standard input (teclado), etc.

Podemos observar a utilização de streams na classe Player.

É utilizado um PrintWriter quando é feito o save das estatísticas de um jogador (utilizado no método saveStatistics).

É utilizado um BufferedReader quando é feito o load das estatísticas de um jogador (utilizado no método loadStatistics).

Programação Event Driven

Um evento é um objeto que representa qualquer ação, quer seja o click de um rato, o premir de uma tecla, entre outras. Quando um componente gera um evento, diz-se que dispara o evento, um componente que dispara um evento pode ter um ou mais objetos listeners, que são usados para tratar dos eventos. Neste trabalho foi utilizada a programação event driven na classe BubbleGameFrame, isto permitiu-nos criar um programa mais funcional e que reage dinamicamente aos pedidos do utilizador. Alguns exemplos disto são os seguintes:

O método implementado para guardar as estatísticas dos jogadores é chamado quando é usado o botão de fecho da janela, isto foi feito com um WindowListener.

Todos os botões e itens dos menus foram realizados com ActionListeners e ActionEvents.