2023

Jogo TicTacToe

EDUARDO ARAÚJO, 30008290
JOÃO LUCAS, 30008215
PEDRO AMARAL, 30008241

Projeto POO: Jogo TicTacToe

Este projeto baseou-se na criação do jogo do galo (TicTacToe) em linguagem de programação Java e com recurso à GUI Java.Swing.

Requisitos:

Visto que o projeto foi desenvolvido por 3 elementos, o requisito foi da possibilidade de poderem estar em jogo até 4 jogadores em simultâneo (humanos e/ou IA).

Funcionalidades:

Com base no requisito do projeto, que se encontra aplicada, foram também desenvolvidas funcionalidades que complementam esse requisito. O conjunto de *features* do programa são as seguintes:

- Até 4 jogadores em simultâneo;
- Seleção dos tipos humano e IA (A3 *Random*) para cada jogador;
- Tabela de jogo com tamanho adaptável;
- Tamanho adaptável para a sequência de linha para pontuar.

Design e Código:

Foi verificado desde cedo no desenvolvimento do projeto que, devido à quantidade de opções do jogo, seria necessário o desenvolvimento de duas interfaces: Interface de configuração do jogo e Interface do jogo. Para tal foram configurados 3 ficheiros para as 3 classes que iriamos utilizar:

- Main.java;
- ConfigFrame.java;
- GameFrame.java.

A relação entre as classes pode ser verificada no diagrama de classes do programa.


Main
Main()
main(String[]) void
ConfigFrame
ConfigFrame()
numGridLabel JLabel
numGridSlider JSlider
numPlayersSlider JSlider
selectedPlayers List
numTypePlayersPane JPanel
typeLabel JLabel
winLabel JLabel
gridPanel JPanel
playerOption String[]
playerComboBox4 JComboBox
normalButton JRadioButton
misereButton JRadioButton
winNumber int
numWinSlider JSlider
numPlayerLabel JLabel
gameOptionPane JPanel
numWinLabel JLabel
sizeOptionPane JPanel
numPlayersPanel JPanel
startButton JButton
typePlayersPanel JPanel
randomButton JRadioButton
playerComboBox3 JComboBox
gridLabel JLabel
frame JFrame
playerNumber int
playerComboBox2 JComboBox
gridSize int
radioButtonGroup ButtonGroup
winPanel JPanel
startPanel JPanel
playerComboBox1 JComboBox
actionPerformed(ActionEvent) void
stateChanged(ChangeEvent) void
«create»
«create»
«create»
GameFrame
GameFrame(int, int, int, List)
currentPlayer String
gameBoardPanel JPanel
cancelPanel JPanel
radioButtonGroup ButtonGroup
playerInfoPanel JPanel
player4Label JLabel
cancelButton JButton
resetButton JButton
radioButtonPane JPanel
scoreSize int
player2ScoreLabel JLabel
playerButton JRadioButton[]
player1ScoreLabel JLabel
playerScoreLabel JLabel[]
scorePanel JPanel
gridLabel JLabel[][]
iconGridMap HashMap<String, String>
frame JFrame
player4ScoreLabel JLabel
currentPlayerScorePane JPanel
player2Label JLabel
currentPlayerLabel JLabel
humanPlayers boolean
availableLabels List
player3Label JLabel
player1Label JLabel
scoreLabel JLabel
player3ScoreLabel JLabel
actionPerformed(ActionEvent) void
mousePressed(MouseEvent) void
verifyColumnScored(JLabel) boolean
verifyPDiagScored(JLabel) boolean
resetGrid() void
mouseReleased(MouseEvent) void
botPlay(ArrayList) JLabel
verifySDiagScored(JLabel) boolean
mouseClicked(MouseEvent) void
mouseEntered(MouseEvent) void
changeSequenceCold(ArrayList<JLabel>, JLabel) void
mouseExited(MouseEvent) void
gameTurn(JLabel) void
verifyRowScored(JLabel) boolean
Powered by yFiles

Main.java

Contém a classe Main() que irá iniciar o programa. Esta classe tem apenas a função de chamar a classe ConfigFrame() que irá gerar a janela de configurações do programa.

```java
public class Main {
    public static void main(String[] args) {

        ConfigFrame configFrame = new ConfigFrame();

    }
}
```

ConfigFrame.java

Classe das configurações de jogo. Esta classe apresenta uma janela ao utilizador onde o mesmo poderá definir as configurações do próximo jogo como, definir o número de jogadores que irão participar e o seu tipo, tamanho da tabela de jogo e o tamanho da sequência da linha para pontuar.

ConfigFrame.java – Módulos Importados

Para que fosse possível criar certos métodos e funções, foi necessário importar módulos utilizados pela GUI (Javax.Swing) e por métodos das interfaces, ActionListener e ChangeListener, implementadas pela classe ConfigFrame(). Foram também importados módulos para utilização de List e ArrayList.

```java
import javax.swing.*;
import javax.swing.event.ChangeEvent;
import javax.swing.event.ChangeListener;
import java.awt.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.util.ArrayList;
import java.util.List;
```

ConfigFrame.java – Variáveis

Na imagem podemos verificar as variáveis declaradas na classe que serão necessárias para a construção da GUI. A quantidade de variáveis deve-se ao facto de cada uma estar associada a um objeto da GUI.

```
12 usages
JFrame frame = new JFrame();
7 usages
JPanel numTypePlayersPanel;
7 usages
JPanel sizeOptionPanel;
8 usages
JPanel gameOptionPanel;
5 usages
JPanel startPanel;
5 usages
JButton startButton;
7 usages
JPanel numPlayersPanel;
10 usages
JPanel typePlayersPanel;
13 usages
JSlider numPlayersSlider;
5 usages
JLabel numPlayerLabel;
2 usages
JLabel gridLabel;
2 usages
JLabel winLabel;
5 usages
JLabel numWinLabel;
5 usages
JLabel numGridLabel;
2 usages
JLabel typeLabel;
11 usages
JSlider numGridSlider;
10 usages
JSlider numWinSlider;
3 usages
JComboBox playerComboBox1;
```

```
3 usages
JComboBox playerComboBox2;
9 usages
JComboBox playerComboBox3;
8 usages
JComboBox playerComboBox4;
8 usages
JPanel gridPanel;
8 usages
JPanel winPanel;
4 usages
JRadioButton normalButton;
4 usages
JRadioButton misereButton;
4 usages
JRadioButton randomButton;
3 usages
int playerNumber;
3 usages
int gridSize;
3 usages
int winNumber;
5 usages
String[] playerOption;
7 usages
List selectedPlayers;
4 usages
ButtonGroup radioButtonGroup;
```

ConfigFrame.java – Método Construtor

Construtor da classe ConfigFrame. Aqui foram definidos valores para as variáveis declaradas na classe e definidas as características dos objetos da GUI. Seguem excertos do código do método construtor e da explicação do mesmo.

No início do método construtor, o programa é configurado para fechar quando o utilizador clicar no botão de fechar a janela e colocar o layout a null para podermos indicar as coordenadas, x e y, de onde queremos cada objeto na JFrame.

```
ConfigFrame(){

    frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    frame.setLayout(null);
```

Atribuição dos valores das variáveis da classe ConfigFrame.

```
numTypePlayersPanel = new JPanel();
numPlayersPanel = new JPanel();
typePlayersPanel = new JPanel();
sizeOptionPanel = new JPanel();
gameOptionPanel = new JPanel();
startPanel = new JPanel();
startButton = new JButton( text: "Start");
numPlayersSlider = new JSlider( min: 2,  max: 4,  value: 2);
numPlayerLabel = new JLabel( text: "2", SwingConstants.CENTER);
gridLabel = new JLabel( text: "Grid", SwingConstants.CENTER);
winLabel = new JLabel( text: "Line Size", SwingConstants.CENTER);
numWinLabel = new JLabel( text: "3");
numGridLabel = new JLabel( text: "3x3", SwingConstants.CENTER);
numGridSlider = new JSlider( min: 3,  max: 9,  value: 3);
numWinSlider = new JSlider( min: 3,  max: 9,  value: 3);
typeLabel = new JLabel( text: "Type", SwingConstants.CENTER);
playerOption = new String[] {"Human", "A3 Random"};
playerComboBox1 = new JComboBox(playerOption);
playerComboBox2 = new JComboBox(playerOption);
playerComboBox3 = new JComboBox(playerOption);
playerComboBox4 = new JComboBox(playerOption);
gridPanel = new JPanel();
winPanel = new JPanel();
normalButton = new JRadioButton( text: "Normal");
misereButton = new JRadioButton( text: "Misère");
randomButton = new JRadioButton( text: "Random Turn");
playerNumber = 2;
gridSize = 3;
winNumber = 1;
selectedPlayers = new ArrayList();
radioButtonGroup = new ButtonGroup();
```

Definição do JPanel com o título "Players" e com o layout "BorderLayout" onde serão colocados mais dois JPanels encostados à esquerda e à direita (WEST e EAST) do painel.

```
numTypePlayersPanel.setBounds( x: 0,  y: 0,  width: 300,  height: 300);
numTypePlayersPanel.setBorder(BorderFactory.createTitledBorder("Players"));
numTypePlayersPanel.setLayout(new BorderLayout( hgap: 10,  vgap: 10));
```

Sub-painel dentro do JPanel "numTypePlayersPanel" onde iremos colocar um JSlider para seleção do número de jogadores que irão participar no jogo e uma JLabel que irá indicar o número de jogadores selecionados no JSlider

```
numPlayersPanel.setPreferredSize(new Dimension( width: 125,  height: 150));
numPlayersPanel.setLayout(new BorderLayout());
numPlayersPanel.setBorder(BorderFactory.createLineBorder(Color.DARK_GRAY));
```

JLabel com o número de jogadores selecionados no JSlider. O texto definido nesta JLabel será atualizado na função stateChanged()

```
numPlayerLabel.setFont(new Font( name: "Consolas", Font.BOLD,  size: 20));
numPlayerLabel.setPreferredSize(new Dimension( width: 100,  height: 75));
```

Configuração da aparência do JSlider e atribuição de ChangeListener para atualização da JLabel numPlayerLabel sempre que o utilizador alterar o JSlider. O valor mínimo de 2 e o valor máximo de 4 do JSlider foram definidos na declaração da variável.

```
numPlayersSlider.setPaintTicks(true);
numPlayersSlider.setPaintTrack(true);
numPlayersSlider.setOrientation(SwingConstants.VERTICAL);
numPlayersSlider.setMajorTickSpacing(1);
numPlayersSlider.addChangeListener( l: this);
```

Sub-painel que irá conter 4 JComboBox onde o utilizador irá indicar se os jogadores serão humanos ou bots.

```
typePlayersPanel.setPreferredSize(new Dimension( width: 160,  height: 150));
typePlayersPanel.setBorder(BorderFactory.createLineBorder(Color.DARK_GRAY));
```

Definição da Layout "GridLayout" para facilitar a organização dos objetos no JPanel typePlayersPanel. A grid terá apenas uma coluna e 5 linhas, onde será colocado um objeto em cada linha com uma distância de 10px entre cada um

```
typePlayersPanel.setLayout(new GridLayout( rows: 5,  cols: 1,  hgap: 10,  vgap: 10));
```

Colocar as JComboBox dos jogadores 3 e 4 desabilitadas ao iniciarmos o programa, visto que o JSlider com o número de jogadores inicia com o valor 2.

```
playerComboBox3.setEnabled(false);
playerComboBox4.setEnabled(false);
```

JPanel com o título 'Size' que irá conter os objetos para o utilizador configurar o tamanho da tabela de jogo e o tamanho da linha necessário para cada jogador pontuar.

```
sizeOptionPanel.setBounds( x: 300,  y: 0,  width: 300,  height: 300);
sizeOptionPanel.setBorder(BorderFactory.createTitledBorder("Size"));
sizeOptionPanel.setLayout(new BorderLayout());
```

Sub-painel que irá conter o JSlider que define o tamanho da tabela de jogo e uma JLabel com a indicação do valor selecionado no JSlider.

```
gridPanel.setPreferredSize(new Dimension( width: 150,  height: 150));
gridPanel.setBorder(BorderFactory.createLineBorder(Color.darkGray));
gridPanel.setLayout(new BorderLayout());
```

JLabel que indica o valor selecionado no JSlider. Definido o tipo de letra "Consolas", a negrito, com o tamanho de letra 25.

```
numGridLabel.setFont(new Font( name: "Consolas", Font.BOLD,  size: 25));
numGridLabel.setPreferredSize(new Dimension( width: 100,  height: 100));
```

JSlider que define o tamanho da tabela de jogo. Slider colocado na vertical com o método setOrientation() e adicionado um "ChangeListener" para que a JLabel numGridLabel seja atualizada com o valor selecionado pelo utilizador no JSlider.

```
numGridSlider.setPaintTicks(true);
numGridSlider.setPaintTrack(true);
numGridSlider.setOrientation(SwingConstants.VERTICAL);
numGridSlider.setMajorTickSpacing(1);
numGridSlider.addChangeListener( l: this);
```

Sub-painel que irá conter os objetos necessários para definirmos o tamanho da linha para pontuar.

```
winPanel.setPreferredSize(new Dimension( width: 125,  height: 125));
winPanel.setBorder(BorderFactory.createLineBorder(Color.darkGray));
winPanel.setLayout(new BorderLayout());
```

JSlider que irá definir o tamanho de linha necessário para um jogador pontuar. Adicionado um ChangeListener que atualiza o valor do JLabel que apresenta o valor selecionado no JSlider.

```
numWinSlider.setPaintTicks(true);
numWinSlider.setPaintTrack(true);
numWinSlider.setOrientation(SwingConstants.VERTICAL);
numWinSlider.setMajorTickSpacing(1);
numWinSlider.addChangeListener( l: this);
```

JLabel que apresenta o valor selecionado no JSlider.

```
numWinLabel.setFont(new Font( name: "Consolas", Font.BOLD,  size: 25));
numWinLabel.setPreferredSize(new Dimension( width: 50,  height: 100));
```

JPanel, com o título Game Options, que irá conter os JRadioButton para seleção do modo de jogo.

```
gameOptionPanel.setBounds( x: 0,  y: 300,  width: 300,  height: 300);
gameOptionPanel.setBorder(BorderFactory.createTitledBorder("Game Options"));
gameOptionPanel.setLayout(new GridLayout( rows: 3,  cols: 1,  hgap: -50,  vgap: -50));
```

Adicionar JRadioButton de cada modo de jogo a um JRadioButtonGroup para que apenas um dos botões possa estar selecionado.

```
radioButtonGroup.add(normalButton);
radioButtonGroup.add(misereButton);
radioButtonGroup.add(randomButton);
```

Mostrar modo normal selecionado ao iniciarmos o jogo. Restantes modos não estão disponíveis.

```
normalButton.setSelected(true);
misereButton.setEnabled(false);
randomButton.setEnabled(false);
```

JPanel para botão de iniciar o jogo.

```
startPanel.setBounds( x: 300,  y: 300,  width: 300,  height: 300);
startPanel.setLayout(null);
```

Botão iniciar com ActionListener para inicializar a *frame* do jogo.

```
startButton.setBounds( x: 75,  y: 120,  width: 150,  height: 50);
startButton.addActionListener( l: this);
```

Adicionar objetos ao JPanel gameOptionPanel.

```
gameOptionPanel.add(normalButton);
gameOptionPanel.add(misereButton);
gameOptionPanel.add(randomButton);
```

Adicionar objetos ao JPanel winPanel.

```
winPanel.add(numWinLabel, BorderLayout.EAST);
winPanel.add(numWinSlider, BorderLayout.WEST);
winPanel.add(winLabel, BorderLayout.NORTH);
```

Adicionar objetos ao JPanel gridPanel.

```
gridPanel.add(numGridLabel, BorderLayout.EAST);
gridPanel.add(numGridSlider, BorderLayout.WEST);
gridPanel.add(gridLabel, BorderLayout.NORTH);
```

Adicionar objetos ao JPanel sizeOptionPanel.

```
sizeOptionPanel.add(winPanel, BorderLayout.EAST);
sizeOptionPanel.add(gridPanel, BorderLayout.WEST);
```

Adicionar objetos ao JPanel typePlayersPanel.

```
typePlayersPanel.add(typeLabel);
typePlayersPanel.add(playerComboBox1);
typePlayersPanel.add(playerComboBox2);
typePlayersPanel.add(playerComboBox3);
typePlayersPanel.add(playerComboBox4);
```

Adicionar objetos ao JPanel numPlayersPanel.

```
numPlayersPanel.add(numPlayerLabel, BorderLayout.NORTH);
numPlayersPanel.add(numPlayersSlider, BorderLayout.SOUTH);
```

Adicionar objetos ao JPanel numTypePlayersPanel.

```
numTypePlayersPanel.add(numPlayersPanel, BorderLayout.WEST);
numTypePlayersPanel.add(typePlayersPanel, BorderLayout.EAST);
```

Adicionar objetos ao JPanel startPanel e os JPanels à JFrame.

```
startPanel.add(startButton);

frame.add(startPanel);
frame.add(gameOptionPanel);
frame.add(sizeOptionPanel);
frame.add(numTypePlayersPanel);
frame.setTitle("Game Configuration");
frame.setSize( width: 610,  height: 620);
frame.setResizable(false);
frame.setBackground(Color.lightGray);
frame.setVisible(true);
```

ConfigFrame.java – Método actionPerformed()

O método actionPerformed() é um método da interface ActionListener que faz com que seja possível codificar funções aos objetos da GUI. Na classe ConfigFrame, este método será apenas chamado quando o utilizador clica no botão *Start* (startButton) e está implementado com o seguinte código:

```
@Override
public void actionPerformed(ActionEvent e) {
    if(e.getSource()==startButton){
        frame.dispose();
        playerNumber = numPlayersSlider.getValue();
        gridSize = numGridSlider.getValue();
        winNumber = numWinSlider.getValue();

        selectedPlayers.add(playerComboBox1.getSelectedItem());
        selectedPlayers.add(playerComboBox2.getSelectedItem());
        if(playerComboBox3.isEnabled() && playerComboBox4.isEnabled()){
            selectedPlayers.add(playerComboBox3.getSelectedItem());
            selectedPlayers.add(playerComboBox4.getSelectedItem());
        } else if(playerComboBox3.isEnabled()){
            selectedPlayers.add(playerComboBox3.getSelectedItem());
        }

        GameFrame gameFrame = new GameFrame(playerNumber, gridSize, winNumber, selectedPlayers);
    }
}
```

No método começa por validar de que objeto foi chamado e validamos se foi chamado pelo startButton. Caso a validação seja verdadeira, a janela com as configurações do jogo é fechada e os valores definidos nos *sliders* são guardados em variáveis. O tipo do primeiro e segundo jogador são também guardados na variável 'selectedPlayers', do tipo 'List'. Depois é validado se as *combo boxes* do terceiro e quarto jogador estão ativas e, caso também estejam ativas, os valores selecionados nessas combo boxes são guardados na variável 'selectedPlayers'.

No final do método, é declarada a variável 'gameFrame' com o método construtor GameFrame que recebe como parâmetros de entrada, as variáveis com os valores das configurações do jogo.

ConfigFrame.java – Método stateChanged()

O método stateChanged() é um método da interface ChangeListener que faz com que seja possível codificar funções aos objetos da GUI. Na classe ConfigFrame, este método irá correr sempre que o utilizador interaja com os *sliders* para selecionar o número de jogadores, o tamanho da tabela de jogo ou o tamanho da sequência para pontuar.

```
    @Override
    public void stateChanged(ChangeEvent e) {
        if(e.getSource() == numPlayersSlider){
            numPlayerLabel.setText(String.valueOf(numPlayersSlider.getValue()));
            if(numPlayersSlider.getValue() == 3){
                playerComboBox3.setEnabled(true);
                playerComboBox4.setEnabled(false);
            } else if(numPlayersSlider.getValue() == 4){
                playerComboBox4.setEnabled(true);
            } else if(numPlayersSlider.getValue() == 2){
                playerComboBox3.setEnabled(false);
                playerComboBox4.setEnabled(false);
            }
        } else if(e.getSource() == numGridSlider){
            numGridLabel.setText(numGridSlider.getValue() + "x" + numGridSlider.getValue());
        } else if(e.getSource() == numWinSlider){
            numWinLabel.setText(String.valueOf(numWinSlider.getValue()));
        }
    }
}
```

O método começa por validar se foi chamado com a alteração do estado do *slider* de seleção do número de jogadores. Caso a validação retorne *true,* o valor da respetiva JLabel é atualizada para o valor selecionado e é ativado o número de combo boxes equivalente para seleção do tipo de jogador.

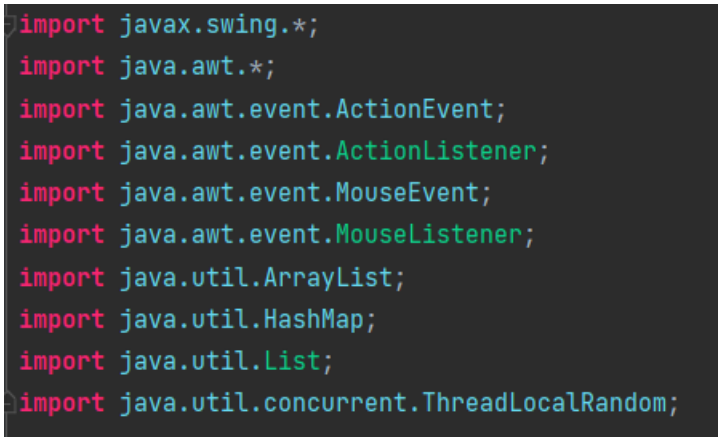
Caso o método tenha sido chamado pelos *sliders* do tamanho da tabela ou da sequência, é atualizada apenas a JLabel respetiva ao *slider* alterado.

GameFrame.java

Classe que contém a GUI do jogo, incluindo a tabela de jogo, jogadores em jogo e pontuação. Esta classe é chamada assim que o botão *Start* é clicado na GUI ConfigFrame e o conteúdo é gerado com base na informação enviada pelas configurações da ConfigFrame. A classe GameFrame contém também todos os métodos necessários para a verificação do jogo do galo, como a verificação se um jogador pontuou ou se a tabela de jogo se encontra completamente preenchida, e jogadas automáticas caso esteja um jogador IA em jogo.

GameFrame.java – Módulos Importados

Como na classe ConfigFrame, para a classe GameFrame foi também necessário importar os módulos de Javax.Swing e da interface ActionListener. Foi ainda necessário importar módulos para MouseListener e Java.Util para utilização de funcionalidades como HashMaps, Lists e ArrayLists.



GameFrame.java – Variáveis

Para a classe GameFrame foi novamente necessário criar uma variável para cada objeto da GUI (do tipo JFrame, JPanel, JLabel, JButton e JRadioButton) e ainda variáveis do tipo String, int, List, boolean e HashMap para guardarmos dados durante o jogo. O modificador de acesso das variáveis não foi definido, o que faz com que todas elas estejam disponíveis para a classe e package.

```
JFrame frame = new JFrame();
5 usages
JPanel gameBoardPanel;
7 usages
JPanel playerInfoPanel;
7 usages
JPanel cancelPanel;
3 usages
JLabel currentPlayerLabel;
3 usages
JLabel scoreLabel;
1 usage
JLabel player1Label;
1 usage
JLabel player2Label;
1 usage
JLabel player3Label;
1 usage
JLabel player4Label;
1 usage
JLabel player1ScoreLabel;
1 usage
JLabel player2ScoreLabel;
1 usage
JLabel player3ScoreLabel;
1 usage
JLabel player4ScoreLabel;
5 usages
JButton cancelButton;
25 usages
JRadioButton[] playerButton;
18 usages
JLabel[] playerScoreLabel;
```

```
61 usages
JLabel[][] gridLabel;
1 usage
JPanel currentPlayerScorePanel;
6 usages
JPanel radioButtonPanel;
6 usages
JPanel scorePanel;
5 usages
String currentPlayer;
2 usages
ButtonGroup radioButtonGroup;
8 usages
JButton resetButton;
13 usages
int scoreSize;
21 usages
HashMap<String, String> iconGridMap;
11 usages
List availableLabels;
3 usages
boolean humanPlayers;
```

GameFrame.java – Método Construtor

Método para criação de um objeto do tipo GameFrame. Neste método são atribuídos valores a todas variáveis que serão utilizadas para a GUI e a sua construção. As configurações dos objetos incluem a sua posição na *frame* do jogo, as suas dimensões e informação a apresentar ao utilizador (texto, imagens, etc).

```
GameFrame(int playerNumber, int gridSize, int scoreNumber, List selectedPlayers){
```

No início do método construtor, configuramos o programa para fechar quando o utilizador clicar no botão de fechar a janela e colocamos o layout a null para podermos indicar as coordenadas, x e y, de onde queremos cada objeto na JFrame.

```
frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
frame.setLayout(null);
```

Atribuição dos valores das variáveis da classe ConfigFrame.

```
gameBoardPanel = new JPanel();
playerInfoPanel = new JPanel();
cancelPanel = new JPanel();
currentPlayerLabel = new JLabel( text: "<html><br>Current<br>Player", SwingConstants.CENTER);
scoreLabel = new JLabel( text: "Score", SwingConstants.CENTER);
player1Label = new JLabel();
player2Label = new JLabel();
player3Label = new JLabel();
player4Label = new JLabel();
player1ScoreLabel = new JLabel( text: "0", SwingConstants.RIGHT);
player2ScoreLabel = new JLabel( text: "0", SwingConstants.RIGHT);
player3ScoreLabel = new JLabel( text: "0", SwingConstants.RIGHT);
player4ScoreLabel = new JLabel( text: "0", SwingConstants.RIGHT);
cancelButton = new JButton( text: "Cancel");
playerButton = new JRadioButton[playerNumber];
gridLabel = new JLabel[gridSize][gridSize];
playerScoreLabel = new JLabel[playerNumber];
currentPlayerScorePanel = new JPanel();
radioButtonPanel = new JPanel();
scorePanel = new JPanel();
radioButtonGroup = new ButtonGroup();
resetButton = new JButton( text: "Reset Grid");
scoreSize = scoreNumber;
iconGridMap = new HashMap<>();
availableLabels = new ArrayList();
humanPlayers = false;
```

JPanel que irá conter a tabela do jogo e definido com o layout GridLayout para o correto posicionamento das JLabels, conforme o tamanho definido pelo utilizador no slider numGridSlider da classe ConfigFrame. Cada JLabel será uma posição onde o jogador poderá efetuar uma jogada. A variável 'gridSize', que contém informação recebida da ConfigFrame, é utilizada para definir o número de linhas e colunas do *Grid Layout* para que a tabela de jogo tenha as dimensões indicadas pelo utilizador.

```
gameBoardPanel.setBounds( x: 25,  y: 25,  width: 500,  height: 600);
gameBoardPanel.setLayout(new GridLayout(gridSize, gridSize));
```

Ciclo 'for' dentro de outro ciclo 'for' para a criação de uma matriz de JLabels, onde cada JLabel será uma posição da tabela do jogo. Para facilitar a identificação das JLabels, o texto definido em cada uma é a sua posição (x e y) na matriz de JLabels, desta forma foi possível detetar que JLabel foi clicada e executar funções sobre essa mesma JLabel. O texto definido é também definido com o tamanho 0 para que não esteja visível para o utilizador.

Ainda neste ciclo 'for', é adicionado um MouseListener, a cada JLabel, para utilização do método mousePressed. Este método fará com que seja possível detetar um clique do utilizador nas JLabels e executar funções, dependendo do jogador que fez a jogada.

```
for(int i = 0; i < gridSize; i++){
    for(int j = 0; j < gridSize; j++){
        gridLabel[i][j] = new JLabel( text: i + " " + j, JLabel.CENTER);
        gridLabel[i][j].setBorder(BorderFactory.createLineBorder(Color.BLACK));
        gridLabel[i][j].setBackground(Color.WHITE);
        gridLabel[i][j].setOpaque(true);
        gridLabel[i][j].setFont(new Font( name: null, Font.PLAIN,  size: 0));

        gridLabel[i][j].addMouseListener( l: this);

        availableLabels.add(gridLabel[i][j].getText());

        gameBoardPanel.add(gridLabel[i][j]);
    }
}
```

JPanel onde será apresentada informação sobre os jogadores em jogo, os seus pontos e que jogador tem a próxima jogada.

```
playerInfoPanel.setBounds( x: 550,  y: 25,  width: 195,  height: 295);
playerInfoPanel.setBorder(BorderFactory.createLineBorder(Color.darkGray));
playerInfoPanel.setLayout(new BorderLayout());
```

JPanel onde serão colocados os JRadioButtons com indicação dos jogadores em jogo e qual deve ser o próximo a jogar. Este JPanel tem o *Grid Layout* definido com 5 linhas e uma coluna onde, em cada linha, será colocado um JRadioButton com cada jogador em jogo.

```
radioButtonPanel.setPreferredSize(new Dimension( width: 100,  height: 50));
radioButtonPanel.setLayout(new GridLayout( rows: 5,  cols: 1,  hgap: -5,  vgap: -5));
```

Apesar de maior parte dos objetos serem adicionados em JPanels e à JFrame no final do método construtor, no caso da JLabel 'currentPlayerLabel' foi necessário adicioná-la ao JPanel 'radioButtonPanel' para que a mesmo seja apresentada acima dos JRadioButtons que serão adicionados mais à frente.

```
currentPlayerLabel.setPreferredSize(new Dimension( width: 70,  height: 50));
radioButtonPanel.add(currentPlayerLabel);
```

JPanel onde serão apresentados os pontos de cada jogador.

```
scorePanel.setPreferredSize(new Dimension( width: 70,  height: 50));
scorePanel.setLayout(new FlowLayout(FlowLayout.CENTER,  hgap: 10,  vgap: 10));
```

JLabel scoreLabel adicionada já ao JPanel scorePanel para que seja apresentado acima das JLabels que irão mostrar os pontos de cada jogador.

```
scoreLabel.setPreferredSize(new Dimension( width: 80,  height: 45));
scorePanel.add(scoreLabel);
```

Ciclo for para a criação dinâmica de JRadioButtons e JLabels, conforme o número de jogadores selecionados nas JComboBoxes e no *slider* numPlayersSlider da ConfigFrame. Cada JRadioButton é adicionado à lista de JRadioButtons playerButton, onde o texto de cada JRadioButton terá o número do jogador e o tipo de jogador selecionado nas JComboBoxes na ConfigFrame. Cada objeto estará desabilitado por defeito. A seguir, cada JRadioButton é adicionado a um JRadioButtonGroup, visto que apenas um pode estar selecionado de cada vez.

```
for(int p = 0; p < playerNumber; p++){

    playerButton[p] = new JRadioButton( text: "P" + (p + 1) + " - " + selectedPlayers.get(p));
    playerButton[p].setEnabled(false);

    radioButtonGroup.add(playerButton[p]);
    radioButtonPanel.add(playerButton[p]);
```

Ainda neste ciclo, adicionamos novas JLabels à lista de JLabels playerScoreLabel, cada uma com o texto '0' predefinido. O texto de cada JLabel será atualizado sempre que for detetado que um jogador conseguiu pontuar. A posição de cada JLabel na lista playerScoreLabel é equivalente à posição do seu jogador na lista playerButton.

```
playerScoreLabel[p] = new JLabel( text: "0", JLabel.RIGHT);
playerScoreLabel[p].setBackground(Color.WHITE);
playerScoreLabel[p].setOpaque(true);
playerScoreLabel[p].setForeground(Color.GREEN);
playerScoreLabel[p].setBorder(BorderFactory.createLineBorder(Color.DARK_GRAY));
playerScoreLabel[p].setFont(new Font( name: null, Font.BOLD,  size: 40));
playerScoreLabel[p].setVerticalTextPosition(JLabel.CENTER);
playerScoreLabel[p].setPreferredSize(new Dimension( width: 65,  height: 45));
scorePanel.add(playerScoreLabel[p]);
```

Ao terminarmos o ciclo 'for', o JRadioButton do primeiro jogador estará selecionado por defeito.

```
playerButton[0].setSelected(true);
playerButton[0].setEnabled(true);
```

JPanel para o botão de cancelamento do jogo e reset da tabela do jogo.

```
cancelPanel.setBounds( x: 550,  y: 430,  width: 195,  height: 195);
cancelPanel.setLayout(null);
cancelPanel.setBorder(BorderFactory.createLineBorder(Color.darkGray));
```

Botão de cancelamento do jogo com ActionListener que irá fechar a GameFrame e reabrir a ConfigFrame.

```
cancelButton.setBounds( x: 50,  y: 45,  width: 100,  height: 50);
cancelButton.addActionListener( l: this);
```

Botão de *reset* da tabela de jogo com ActionListener que irá colocar o ícone de cada JLabel da matriz a *null* e colocar, também, cada JLabel na lista 'availableLabels'.

```
resetButton.setBounds( x: 50,  y: 105,  width: 100,  height: 50);
resetButton.addActionListener( l: this);
resetButton.setVisible(false);
```

Adicionar objetos aos respetivos JPanels.

```
playerInfoPanel.add(radioButtonPanel, BorderLayout.WEST);
playerInfoPanel.add(scorePanel, BorderLayout.EAST);


cancelPanel.add(cancelButton);
cancelPanel.add(resetButton);

frame.add(playerInfoPanel);
frame.add(cancelPanel);
frame.add(gameBoardPanel);
frame.setTitle("TicTacToe Game");
frame.setSize( width: 800,  height: 700);
frame.setResizable(false);
frame.setBackground(Color.lightGray);
frame.setVisible(true);
```

Ainda antes do método construtor terminar, validamos a presença de jogadores humanos no jogo. Este ciclo irá verificar se pelo menos um dos JRadioButtons dos jogadores contém a String "Human" onde, caso a condição se verifique, atribui o valor *true* à variável 'humanPlayers'. Caso 'humanPlayers' se mantenha com o valor *false*, o botão de *reset* não será apresentado quando a tabela de jogo estiver preenchida.

```
for(JRadioButton i: playerButton){
    if (i.getText().contains("Human")){
        humanPlayers = true;
    }
}
```

Efetuamos também uma jogada automática após iniciação da 'JFrame', no caso do primeiro jogador ser A3 Random (Bot).

```
if(selectedPlayers.get(0).equals("A3 Random")){
    gameTurn(botPlay((ArrayList)availableLabels));
}
```

GameFrame.java – Método ActionPerformed()

Método executado quando um dos JButtons 'cancelButton' ou 'resetButton' é pressionado. O 'cancelButton' irá fechar a GameFrame e abrir uma nova ConfigFrame, enquanto o 'resetButton' irá correr o método resetGrid e colocar-se invisível.

```
@Override
public void actionPerformed(ActionEvent e) {
    if(e.getSource() == cancelButton){
        frame.dispose();
        ConfigFrame configFrame = new ConfigFrame();
    } else if(e.getSource() == resetButton){
        resetGrid();
        resetButton.setVisible(false);
    }
}
```

GameFrame.java – Método mousePressed()

Método que irá correr quando o rato for pressionado sobre uma das JLabels da matriz de JLabels. Será corrido o método gameTurn que irá receber a JLabel pressionada como parâmetro.

```
@Override
public void mousePressed(MouseEvent e) { gameTurn((JLabel) e.getSource()); }
```

GameFrame.java – Método botPlay()

Método para a jogada automática do Bot A3 Random e será usada como parâmetro para o método gameTurn(). Este método irá selecionar aleatoriamente uma posição da lista 'availableLabels', que contém as coordenadas de todas as JLabels disponíveis e retorna a JLabel correspondente a essas coordenadas. A JLabel selecionada será depois utilizada pelo método gameTurn() que irá efetuar a jogada.

```
public JLabel botPlay(ArrayList availableLabels){
    String[] selectedLabel;
    try {
        selectedLabel = availableLabels.get(ThreadLocalRandom.current().nextInt( origin: 0, availableLabels.size())).toString().split( regex: " ");
    } catch (Exception e){
        selectedLabel = new String[]{"0", "0"};
    }
    return gridLabel[Integer.parseInt(selectedLabel[0])][Integer.parseInt(selectedLabel[1])];
}
```

GameFrame.java – Método gameTurn()

Método de execução da jogada efetuada pelo jogador humano ou bot. O mesmo recebe uma JLabel da matriz de JLabels como parâmetro de entrada e efetua todas as operações necessárias, como alteração de ícones e chamar os métodos de validação de linhas, com base na JLabel clicada.

O método começa por validar se a JLabel clicada não tem um ícone definido. Caso a condição seja verdadeira, a coordenada da JLabel será retirada da lista 'availableLabels' e será iniciado o ciclo for. Caso seja falso, o método irá terminar sem efetuar qualquer outra operação.

```
public void gameTurn(JLabel clickedLabel){
    if(clickedLabel.getIcon() == null){
        availableLabels.remove(clickedLabel.getText());
```

A seguir, validamos que jogador efetuou a jogada, verificando que JRadioButton se encontra ativo. Caso o JRadioButton no índex 0 se encontre ativo, a jogada foi efetuada pelo jogador P1. Será adicionado ao HashMap 'iconGridMap' o mapeamento entre as coordenadas da JLabel selecionada e o caminho da imagem de "X", o ícone da JLabel selecionada será alterado para "X" e o JRadioButton do jogador P1 será desativado.

```
for(int b = 0; b < playerButton.length; b++){

    if(playerButton[b].isSelected()){
        switch (b){
            case 0:
                iconGridMap.put(clickedLabel.getText(), "images\\x_mark.png");
                gridLabel[Integer.parseInt(clickedLabel.getText().split( regex: " ")[0])][Integer.parseInt(clickedLabel.getText().split( regex: " ")[1])].setIcon(new ImageIcon(new
                playerButton[b].setEnabled(false);
                break;
```

Caso o JRadioButton no índex 1 se encontre ativo, a jogada foi efetuada pelo jogador P2. Será adicionado ao HashMap 'iconGridMap' o mapeamento entre as coordenadas da JLabel selecionada e o caminho da imagem de "circulo", o ícone da JLabel selecionada será alterado para "circulo" e o JRadioButton do jogador P2 será desativado.

```
case 1:
    iconGridMap.put(clickedLabel.getText(), "images\\circle_mark.png");
    gridLabel[Integer.parseInt(clickedLabel.getText().split( regex: " ")[0])][Integer.parseInt(clickedLabel.getText().split( regex: " ")[1])].setIcon(new ImageIcon(new
    playerButton[b].setEnabled(false);
    break;
```

Caso o JRadioButton no índex 2 se encontre ativo, a jogada foi efetuada pelo jogador P3. Será adicionado ao HashMap 'iconGridMap' o mapeamento entre as coordenadas da JLabel selecionada e o caminho da imagem de "triangulo", o ícone da JLabel selecionada será alterado para "triangulo" e o JRadioButton do jogador P3 será desativado.

```
case 2:
    iconGridMap.put(clickedLabel.getText(), "images\\triangle.png");
    gridLabel[Integer.parseInt(clickedLabel.getText().split( regex: " ")[0])][Integer.parseInt(clickedLabel.getText().split( regex: " ")[1])].setIcon(new ImageIcon(new
    playerButton[b].setEnabled(false);
    break;
```

Caso o JRadioButton no índex 3 se encontre ativo, a jogada foi efetuada pelo jogador P4. Será adicionado ao HashMap 'iconGridMap' o mapeamento entre as coordenadas da JLabel selecionada e o caminho da imagem de "quadrado", o ícone da JLabel selecionada será alterado para "quadrado" e o JRadioButton do jogador P4 será desativado.

```
case 3:
    iconGridMap.put(clickedLabel.getText(), "images\\square.png");
    gridLabel[Integer.parseInt(clickedLabel.getText().split( regex: " ")[0])][Integer.parseInt(clickedLabel.getText().split( regex: " ")[1])].setIcon(new ImageIcon(new
    playerButton[b].setEnabled(false);
    break;
```

Validar se o jogador pontuou na linha da JLabel selecionada. Caso o método retorne *true*, o texto da JLabel com a pontuação do jogador é atualizado para o valor atual + 1.

```
if(verifyRowScored(clickedLabel)){
    playerScoreLabel[b].setText(String.valueOf( i: Integer.parseInt(playerScoreLabel[b].getText()) + 1));
}
```

Validar se o jogador pontuou na coluna da JLabel selecionada. Caso o método retorne *true*, o texto da JLabel com a pontuação do jogador é atualizado para o valor atual + 1.

```
if(verifyColumnScored(clickedLabel)){
    playerScoreLabel[b].setText(String.valueOf( i: Integer.parseInt(playerScoreLabel[b].getText()) + 1));
}
```

Validar se o jogador pontuou na diagonal principal da JLabel selecionada. Caso o método retorne *true*, o texto da JLabel com a pontuação do jogador é atualizado para o valor atual + 1.

```
if(verifyPDiagScored(clickedLabel)){
    playerScoreLabel[b].setText(String.valueOf( i: Integer.parseInt(playerScoreLabel[b].getText()) + 1));
}
```

Validar se o jogador pontuou na diagonal secundária da JLabel selecionada. Caso o método retorne *true*, o texto da JLabel com a pontuação do jogador é atualizado para o valor atual + 1.

```
if(verifySDiagScored(clickedLabel)){
    playerScoreLabel[b].setText(String.valueOf( i: Integer.parseInt(playerScoreLabel[b].getText()) + 1));
}
```

Verificar se a lista 'availableLabels' se encontra vazia e se existem jogadores humanos em jogo. O JButton 'resetButton' ficará disponível caso ambas as condições sejam verdadeiras.

```
if(availableLabels.size() == 0 && humanPlayers){
    resetButton.setVisible(true);
}
```

Validar quem será o próximo jogador, ativando o JRadioButton que se segue na lista 'playerButton'. Caso o JRadioButton que se encontra ativo seja o último da lista, será ativado o JRadioButton do índex 0 da lista playerButton. Caso o jogador seguinte seja um bot e ainda existam JLabels disponíveis na tabela do jogo, efetuar jogada automática.

```
if(b == playerButton.length - 1){
    playerButton[0].setSelected(true);
    playerButton[0].setEnabled(true);
    currentPlayer = playerButton[0].getText();

    if(currentPlayer.contains("A3 Random") && availableLabels.size() > 0){
        gameTurn(botPlay((ArrayList)availableLabels));
    }
} else {
    playerButton[b+1].setEnabled(true);
    playerButton[b+1].setSelected(true);
    currentPlayer = playerButton[b + 1].getText();
    if(currentPlayer.contains("A3 Random") && availableLabels.size() > 0){
        gameTurn(botPlay((ArrayList)availableLabels));
    }
}
```

GameFrame.java – Método verifyRowScored()

Método de validação da linha da JLabel selecionada. A partir das coordenadas da JLabel, é efetuada *forward lookup* e *backwards lookup*, ambas sequenciais, onde cada uma para apenas quando uma das seguintes situações é detetada:

- É encontrada uma JLabel na linha que não contém um ícone igual ao da JLabel selecionada;

- A pesquisa chega ao início da linha (no caso da *backwards lookup*);

- A pesquisa chega ao final da linha (no caso do *forward lookup*).

No início do método, é criada uma variável do tipo 'int array' com o resultado do *split* do texto da JLabels (as coordenadas x e y da JLabel na matriz). São também declaradas as de estado das pesquisas, onde o valor de uma ou ambas as variáveis será alterado para false caso seja detetada uma ou mais situações indicadas anteriormente, o que fará com que essa pesquisa não seja novamente efetuada durante a restante execução do método. É também declarada uma ArrayList de JLabels que irá conter todas as JLabels encontradas, cujo ícone definido é igual ao ícone da JLabel selecionada.

De seguida é iniciado o ciclo for que irá correr o número de vezes igual ao tamanho da sequencia para pontuar.

```
public boolean verifyRowScored(JLabel clickedLabel) {
    int[] rowColumn = {Integer.parseInt(clickedLabel.getText().split( regex: " ")[0]), Integer.parseInt(clickedLabel.getText().split( regex: " ")[1])};
    boolean fRowState = true;
    boolean bRowState = true;
    ArrayList<JLabel> rowSequence = new ArrayList<>();

    for (int i = 1; i < scoreSize; i++) {
```

No início do ciclo, é efetuado o *forward lookup* da linha. Caso não tenhamos ultrapassado a margem da tabela de jogo e 'fRowState' sejam *true*, é validado se a JLabel na tabela de jogo tem ícone definido e se na HashMap 'iconGridMap', o ícone da coordenada da nova JLabel é igual ao ícone da coordenada da JLabel selecionada. Se sim, a nova JLabel pesquisada é adicionada à lista de jLabels 'rowSequence'.

```
if(rowColumn[1] + i < gridLabel.length && fRowState){
    if(gridLabel[rowColumn[0]][rowColumn[1] + i].getIcon() != null && (iconGridMap.get(gridLabel[rowColumn[0]][rowColumn[1] + i].getText()).equals(
                                                                    iconGridMap.get(clickedLabel.getText())))){
        rowSequence.add(gridLabel[rowColumn[0]][rowColumn[1] + i]);
    } else{
        fRowState = false;
    }
}
```

Backwards lookup da linha. Efetua o mesmo procedimento que a *forward lookup* mas no sentido inverso.

```
if(rowColumn[1] - i >= 0 && bRowState){
    if(gridLabel[rowColumn[0]][rowColumn[1] - i].getIcon() != null && (iconGridMap.get(gridLabel[rowColumn[0]][rowColumn[1] - i].getText()).equals(
                                                                    iconGridMap.get(clickedLabel.getText())))){

        rowSequence.add(gridLabel[rowColumn[0]][rowColumn[1] - i]);
    } else{
        bRowState = false;
    }
}
```

Caso ‘rowSequence’ contenha um valor igual ou maior de JLabels que o definido no slider numWinSlider na ConfigFrame, executar o método changeSequenceColor(). Enviadas as variáveis ‘rowSequence’ e ‘clickedLabel’ como parâmetros de entrada. Retornar *true* ou *false* conforme o resultado da validação entre o tamanho de ‘rowSequence’ e o valor de ‘scoreSize’.

```
if(rowSequence.size() + 1 ≥ scoreSize){
    changeSequenceColor(rowSequence, clickedLabel);
}

return rowSequence.size() + 1 ≥ scoreSize;
```

GameFrame.java – Método verifyColumnScored()

Método com código equivalente a verifyRowScored() mas para validação da coluna onde foi efetuada a jogada.

```
public boolean verifyColumnScored(JLabel clickedLabel){
    int[] rowColumn = {Integer.parseInt(clickedLabel.getText().split( regex: " ")[0]), Integer.parseInt(clickedLabel.getText().split( regex: " ")[1])};
    boolean fColumnState = true;
    boolean bColumnState = true;
    ArrayList<JLabel> columnSequence = new ArrayList<>();

    for (int i = 1; i < scoreSize; i++) {
        if(rowColumn[0] + i < gridLabel.length && fColumnState){
            if(gridLabel[rowColumn[0] + i][rowColumn[1]].getIcon() != null && (iconGridMap.get(gridLabel[rowColumn[0] + i][rowColumn[1]].getText()).equals(
                                                                    iconGridMap.get(clickedLabel.getText())))){
                columnSequence.add(gridLabel[rowColumn[0] + i][rowColumn[1]]);
            } else{
                fColumnState = false;
            }
        }
        if(rowColumn[0] - i ≥ 0 && bColumnState){
            if(gridLabel[rowColumn[0] - i][rowColumn[1]].getIcon() != null && (iconGridMap.get(gridLabel[rowColumn[0] - i][rowColumn[1]].getText()).equals(
                                                                        iconGridMap.get(clickedLabel.getText())))){
                columnSequence.add(gridLabel[rowColumn[0] - i][rowColumn[1]]);
            } else{
                bColumnState = false;
            }
        }
    }
    if(columnSequence.size() + 1 ≥ scoreSize){
        changeSequenceColor(columnSequence, clickedLabel);
    }
    return columnSequence.size() + 1 ≥ scoreSize;
}
```

GameFrame.java – Método verifyPDiagScored()

Método com código equivalente a verifyRowScored() mas para validação da diagonal principal onde foi efetuada a jogada.

```
public boolean verifyPDiagScored(JLabel clickedLabel){
    int[] rowColumn = {Integer.parseInt(clickedLabel.getText().split( regex: " ")[0]), Integer.parseInt(clickedLabel.getText().split( regex: " ")[1])};
    boolean fPDiagState = true;
    boolean bPDiagState = true;
    ArrayList<JLabel> pDiagSequence = new ArrayList<>();

    for (int i = 1; i < scoreSize; i++) {
        if((rowColumn[0] + i < gridLabel.length && rowColumn[1] + i < gridLabel.length) && fPDiagState){
            if(gridLabel[rowColumn[0] + i][rowColumn[1] + i].getIcon() != null && (iconGridMap.get(gridLabel[rowColumn[0] + i][rowColumn[1] + i].getText()).equals(
                                                                                    iconGridMap.get(clickedLabel.getText())))){
                pDiagSequence.add(gridLabel[rowColumn[0] + i][rowColumn[1] + i]);
            } else{
                fPDiagState = false;
            }
        }

        if((rowColumn[0] - i  >= 0 && rowColumn[1] - i >= 0) && bPDiagState){
            if(gridLabel[rowColumn[0] - i][rowColumn[1] - i].getIcon() != null && (iconGridMap.get(gridLabel[rowColumn[0] - i][rowColumn[1] - i].getText()).equals(
                                                                                    iconGridMap.get(clickedLabel.getText())))){
                pDiagSequence.add(gridLabel[rowColumn[0] - i][rowColumn[1] - i]);
            } else{
                bPDiagState = false;
            }
        }
    }

    if(pDiagSequence.size() + 1 >= scoreSize){
        changeSequenceColor(pDiagSequence, clickedLabel);
    }
    return pDiagSequence.size() + 1 >= scoreSize;
}
```

GameFrame.java – Método verifySDiagScored()

Método com código equivalente a verifyRowScored() mas para validação da diagonal secundária onde foi efetuada a jogada.

```
public boolean verifySDiagScored(JLabel clickedLabel){
    int[] rowColumn = {Integer.parseInt(clickedLabel.getText().split( regex: " ")[0]), Integer.parseInt(clickedLabel.getText().split( regex: " ")[1])};
    boolean fSDiagState = true;
    boolean bSDiagState = true;
    ArrayList<JLabel> sDiagSequence = new ArrayList<>();

    for (int i = 1; i < scoreSize; i++) {
        if((rowColumn[0] + i < gridLabel.length && rowColumn[1] - i >= 0) && fSDiagState){
            if(gridLabel[rowColumn[0] + i][rowColumn[1] - i].getIcon() != null && (iconGridMap.get(gridLabel[rowColumn[0] + i][rowColumn[1] - i].getText()).equals(
                                                                                    iconGridMap.get(clickedLabel.getText())))){
                sDiagSequence.add(gridLabel[rowColumn[0] + i][rowColumn[1] - i]);
            } else{
                fSDiagState = false;
            }
        }

        if((rowColumn[0] - i  >= 0 && rowColumn[1] + i < gridLabel.length) && bSDiagState){
            if(gridLabel[rowColumn[0] - i][rowColumn[1] + i].getIcon() != null && (iconGridMap.get(gridLabel[rowColumn[0] - i][rowColumn[1] + i].getText()).equals(
                                                                                    iconGridMap.get(clickedLabel.getText())))){
                sDiagSequence.add(gridLabel[rowColumn[0] - i][rowColumn[1] + i]);
            } else{
                bSDiagState = false;
            }
        }
    }

    if(sDiagSequence.size() + 1 >= scoreSize){
        changeSequenceColor(sDiagSequence, clickedLabel);
    }
    return sDiagSequence.size() + 1 >= scoreSize;
}
```

GameFrame.java – Método resetGrid()

Método executado após ser o botão 'resetButton' ser clicado. Efetuado um ciclo 'for' dentro de outro ciclo 'for' que irão percorrer a matriz de JLabels, alterando os ícones de cada JLabel para *null*. As coordenadas de cada JLabel são também repostas na lista 'availableLabels'. Caso, no momento em que o botão 'resetButton' é pressionado, o próximo jogador seja A3 Random, é efetuada uma jogada automática.

```
public void resetGrid(){
    for(int i = 0; i < gridLabel.length; i++){
        for(int j = 0; j < gridLabel.length; j++){
            gridLabel[i][j].setIcon(null);
            gridLabel[i][j].setBackground(null);
            availableLabels.add(gridLabel[i][j].getText());
        }
    }
    if(currentPlayer.contains("A3 Random")){
        gameTurn(botPlay((ArrayList)availableLabels));
    }
}
```

GameFrame.java – Método changeSequenceColor()

Método para alteração do background das JLabels recebidas pelo parâmetro de entrada 'sequence'. Este método é executado quando é detetada uma sequência de JLabels da tabela de jogo com tamanho igual ou superior ao indicado nas configurações, e com o mesmo ícone definido. As cores serão diferentes dependendo do jogador que efetuou a jogada.

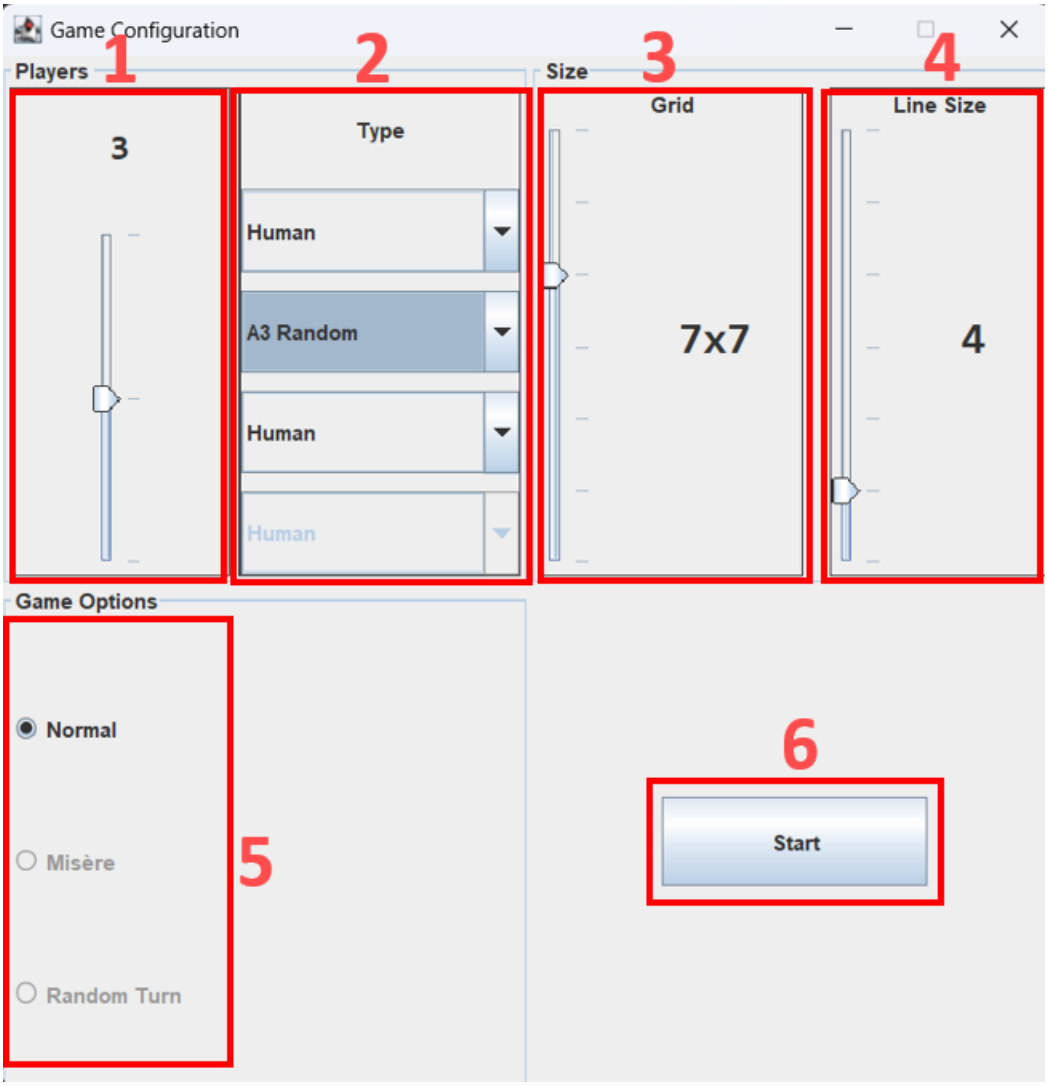
```
public void changeSequenceColor(ArrayList<JLabel> sequence, JLabel clickedLabel){

    for(JLabel i: sequence){
        int[] rowColumn = {Integer.parseInt(i.getText().split( regex: " ")[0]), Integer.parseInt(i.getText().split( regex: " ")[1])};
        if(playerButton[0].isSelected()){
            gridLabel[rowColumn[0]][rowColumn[1]].setBackground(Color.red.darker());
            clickedLabel.setBackground(Color.RED.darker());
        } else if(playerButton[1].isSelected()){
            gridLabel[rowColumn[0]][rowColumn[1]].setBackground(Color.GREEN.darker());
            clickedLabel.setBackground(Color.GREEN.darker());
        } else if(playerButton[2].isSelected()){
            gridLabel[rowColumn[0]][rowColumn[1]].setBackground(Color.BLUE.darker());
            clickedLabel.setBackground(Color.BLUE.darker());
        } else if(playerButton[3].isSelected()) {
            gridLabel[rowColumn[0]][rowColumn[1]].setBackground(Color.YELLOW.darker());
            clickedLabel.setBackground(Color.YELLOW.darker());
        }
    }
}
```

Graphical User Interface

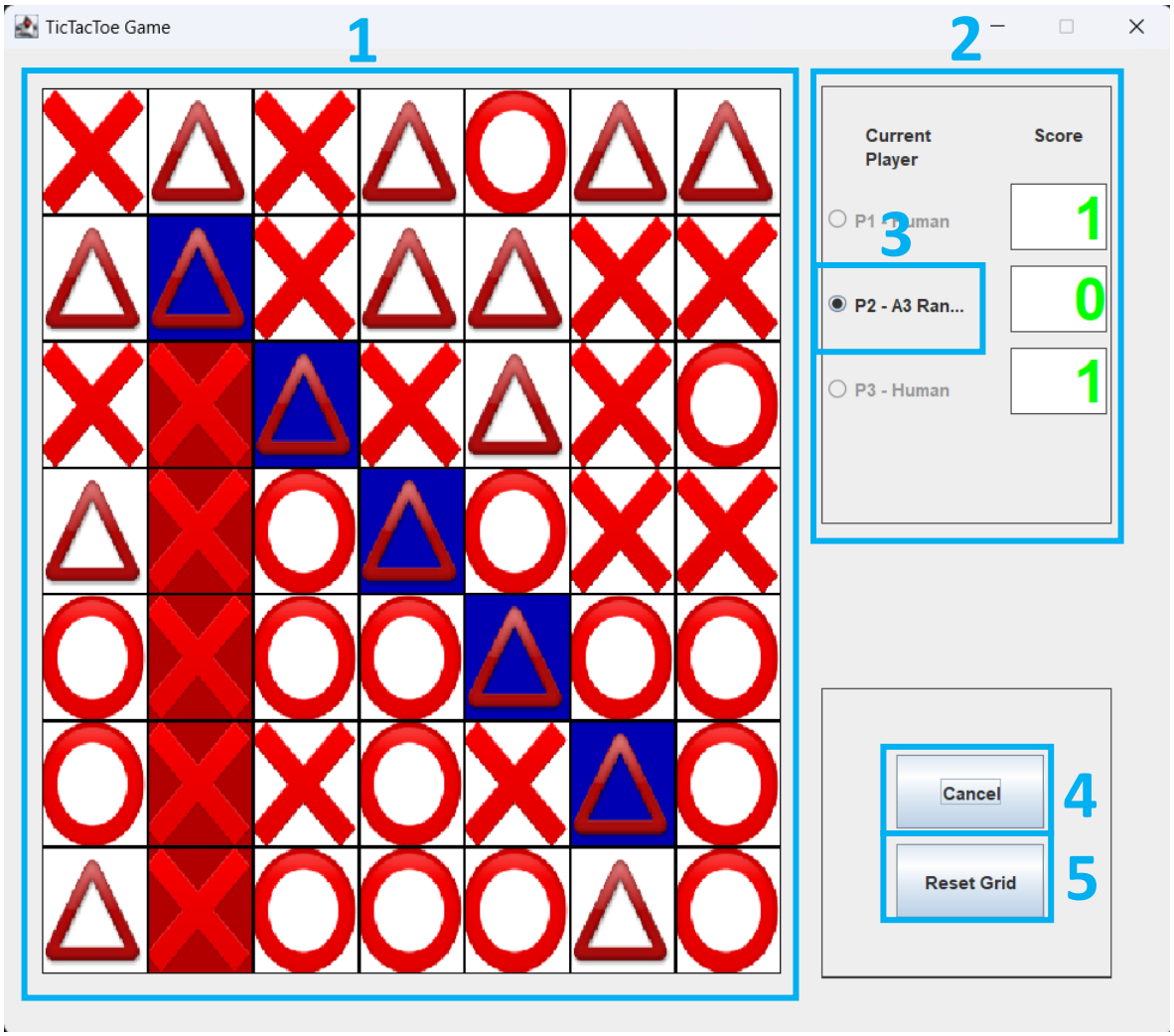
Como já indicado anteriormente, neste projeto foi utilizado Java Swing, uma API que providencia um ambiente gráfico a aplicações java. Swing faz parte da Oracle's Java Foundation Classes (uma framework de construção de GUIs baseadas em Java). Para este projeto optámos por uma interface simplista, baseada na GUI dos exemplos disponibilizados pelo Professor.

Graphical User Interface - ConfigFrame



1) *Slider* de seleção de número de jogadores a participar no jogo
2) *Combo boxes* de seleção do tipo de jogador (humano ou A3 Random) que irá participar no jogo
3) *Slider* de seleção do tamanho da tabela de jogo
4) *Slider* de seleção do tamanho da sequência da linha para pontuar
5) Modos de jogo
6) Butão para iniciar o jogo

Graphical User Interface – GameFrame



1) Tabela de jogo
2) Tabela de jogadores e pontuações
3) Próximo jogador a jogar
4) Butão para cancelar o jogo e voltar à janela de configurações
5) Butão para reiniciar a tabela de jogo (apenas disponível quando a tabela de jogo se encontra preenchida

Informação técnicas e estatísticas

Source File	Total Lines	Source Code ...	Source Code ...	Comment Lin...	Comment Lin...	Blank Lines	Blank Lines [%]
ConfigFrame.java	307	198	64%	59	19%	50	16%
GameFrame.java	573	379	66%	112	20%	82	14%
Main.java	7	5	71%	0	0%	2	29%
Total:	887	582	66%	171	19%	134	15%