IPS Instituto Politécnico de Setúbal
Escola Superior de Tecnologia de Setúbal

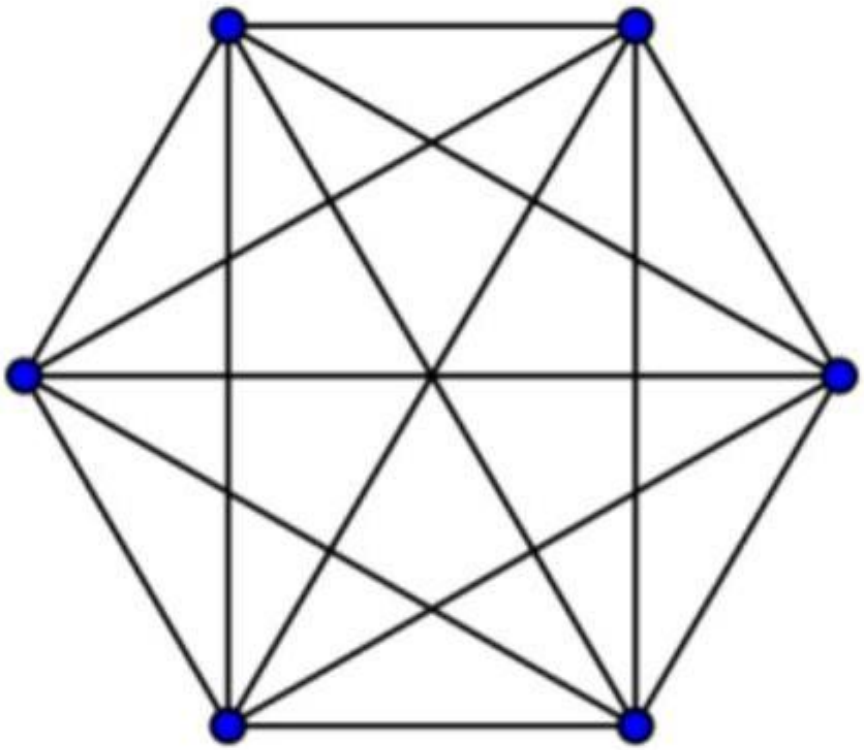
Curso :

Engenharia Informática

Ano Letivo 2017/2018

Relatório Técnico

Jogo SIM(paper-and-pencil)



Relatório realizado por:

-Tiago Neto 160221086

-Hugo Ferreira 160221089

Padrões Implementados

Singleton- Este padrão foi utilizado para elaborar o logger e as configurações do sistema. As configurações de logger poderão ser manipuladas através do respectivo menu de configurações.A utilização de um Singleton deve-se ao facto de necessitarmos apenas de uma instância da classe em causa;

Memento- Utilizado para guardar o estado do tabuleiro ao longo do jogo para que, desta forma, possa ser desfeita uma jogada. A utilização deste padrão justifica-se pela necessidade de guardar e alternar entre os vários estados do tabuleiro de jogo.

MVP- Utilizado em diversas classes de interação com o utilizador de modo a efetuar todas as verificações e validações do input recebido. No geral, utilizámos como View uma classe em Java Fx e como Model uma classe destinada a verificações e interação com as persistência dos dados. A utilização do padrão permite uma forma mais eficaz de definir responsabilidades acabando por simplificar a classe view. Interliga o model e a view de forma a que toda a visualização e inserção de dados seja tratada corretamente.

Adapter- Padrão utilizado nas estatísticas onde é necessário ordenar uma lista de jogadores por ordem de vitórias. Optou-se por adaptar uma LinkedList para uma LinkedListAdapter com a diferença de que, nesta, a inserção de elementos é feita logo por ordem de vitórias.

DAO - Padrão utilizado para a persistência dos dados. Optámos por 3 implementações diferente permitindo, desta forma, 3 tipos diferentes de persistência. Sqlite, json e serializable cuja escolha e alteração pode ser feita através do java persistance file anexado na pasta do projeto. Utilizando o padrão DAO, a alteração de tipo de persistência de dados é simplificada, criando uma nova classe apenas se for necessário.

Observer- Padrão utilizado para dar a ordem de quando devem ser feitos registos no Logger ou alterar as estatísticas dos jogadores quando um jogo inicia e termina. Os observadores são as classes StartGame e FinishGame. A utilização deste padrão permitiu simplificar os métodos start e finish game visto que ambos têm mais funções a executar. Tais como escrever no looger e atualizar as estatísticas.

Factories- Padrão utilizado de forma a simplificar a escolha dos tipos de jogo, bem como o tipo de persistência a utilizar.

Template- Usado para diferenciar o “raciocínio” da máquina durante o jogo na escolha da aresta dependendo, obviamente, da dificuldade de jogo escolhida. A utilização deste padrão ao invés do Strategy deve-se ao facto de seja qual for o nível de dificuldade do jogo, existem informações que necessitam de ser acedidas

pelos vários algoritmos. A utilização de um tipo de padrão que assenta na hierarquia de classes permite um acesso aos dados facilitado sem necessidade do recurso a passar atributos por parâmetro.

Strategy- Usado para definir como a lista de jogadores iria ser organizada, ou por jogos ganhos ou por nome. A utilização deste padrão justifica-se pelo facto de que a necessidade de organizar a lista por outro critério de ordenação, basta ser passado por parâmetro um novo comparador.

TADS utilizadas

TAD Graph - O TAD Graph implementado é utilizado para a implementação do gráfico do tabuleiro de jogo. Utilzámos um grafo porque, desta forma, é-nos permitido interligar arestas com vértices onde neste caso cada um deles pode guardar as informações necessária. No caso dos vértices, as suas coordenadas, e no caso das arestas, o jogador seletor e cor. O conjunto dos dois permite, posteriormente, o seu desenho.

LinkedListAdapter - O LinkedListAdapter é utilizado para as estatísticas onde, como já foi referido anteriormente, é necessário ter uma lista de jogadores ordenados por número de vitórias. Optámos por uma tipo de lista adaptável, para que seja simplificada a alteração do critério de ordenação da lista.

Diagrama de classes

Anexado na pasta do projeto o diagrama UML.

Diagrama de Player DAO

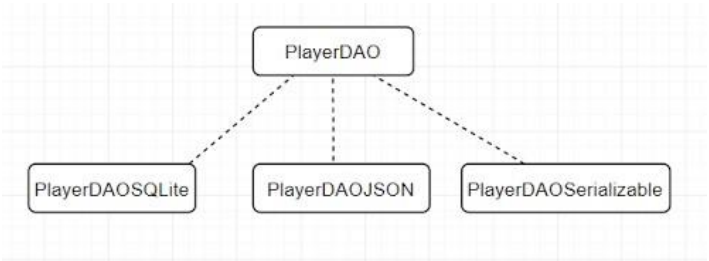


Diagrama de jogadores

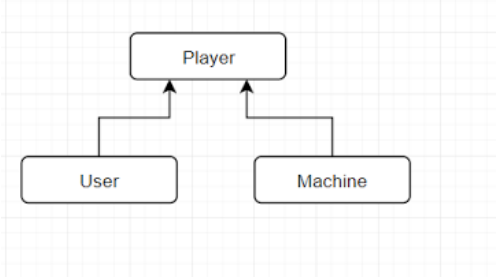
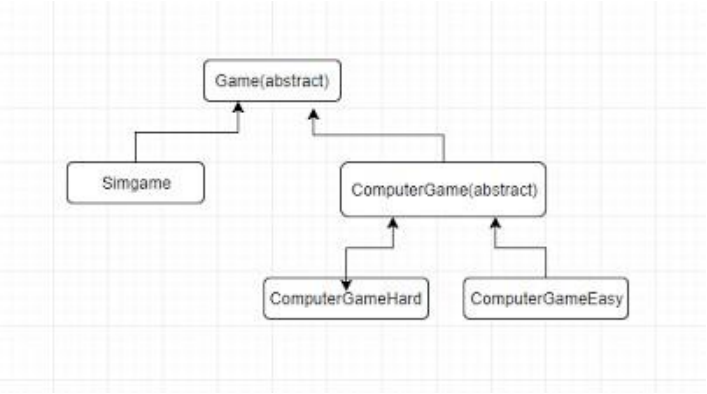


Diagrama de Player DAO



Documentação de classes

Toda a documentação em JavaDoc está anexada na pasta do projeto.

Nesta área será, ainda, feita uma breve descrição das classes mais importantes e essenciais á base de funcionamento do sistema. (Integrantes do package model).

Checker (Superclasse) - Classe destinada à interação entre classes de persistência e validação de dados introduzidos pelo utilizador.

- LoginChecker- Responsável por validações a nível de autenticação de utilizadores.
- RegisterChecker (Superclasse) - Responsável por validações a nível de registo de utilizadores.
    - AccountChecker- Responsável por validações a nível de alterações de contas de utilizadores.

Game (Superclasse) - Classe abstrata que define alguns atributos e métodos de um jogo.

- SimGame- Representa a implementação do tipo de jogo onde se defrontarão dois utilizadores.
- ComputerGame (Superclasse)- Classe abstrata que define alguns atributos e métodos de um jogo em que o adversário é um computador.

    ComputerGameEasy- Representa a implementação do modo de jogo fácil frente ao computador. O raciocínio de escolha das arestas por parte da máquina é feita aleatoriamente, através da implementação de um random.

ComputerGameHard- Representa a implementação do modo de jogo difícil frente ao computador. O raciocínio de escolha das arestas por parte da máquina é feita de forma calculada através da implementação de um algoritmo que pode ser encontrado mais abaixo. (chooseEdge()).

Board- Classe que contém o grafo constituinte da figura de jogo.

Connection- Representa uma aresta do hexágono de jogo.

Corner- Representa um vértice do hexágono de jogo.

IndividualStatistics- Classe que guarda os tipos de estatísticas de cada utilizador por tipo de jogo.

Statistic- Classe que armazena as estatísticas individuais de um jogador. Número total de jogos, vitórias, derrotas, e tempo de jogo.

StatisticsManager - Representa a lista total de estatísticas organizada pela ordem e critério requerido.

Player (Superclasse)- Representa um jogador que tanto pode ser um utilizador como um computador.

- User- Representa um utilizador do sistema.
- Machine - Classe que representa um computador.

PlayersManager - Classe onde se pode encontrar a persistência de dados do sistema e os registos de autenticações efetuadas.

RowData - Representa uma linha da tabela de estatísticas preparada para receber as informações de determinado jogador.

SystemConfiguration - Representa a classe onde será lido e escolhido o tipo de persistência a utilizar no sistema.

Distribuição de classes por respetivos packages

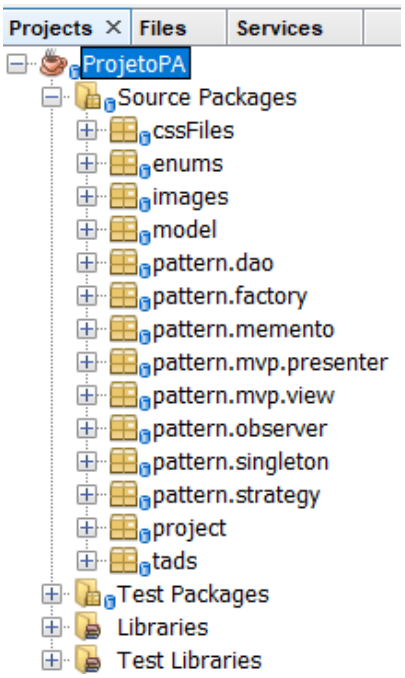
Package cssFiles - Package onde estão os ficheiros de especificação dos layouts implementados.

Package pattern.dao - Package relativo à persistência dos dados do sistema.

Package enums - Classes do tipo enumeradas que permitem a escolha entre várias opções face a determinadas situações.

Package pattern.factory- Package onde estão as implementações do padrão factory para a construção do tipo de persistência e modo de jogo.

Package images- Package onde estão as imagens utilizadas a nível de layout do projeto.



Package pattern.singleton- Implementações do padrão singleton. Logger e configurações.

Package pattern.memento- Implementação do padrão memento utilizado para refazer as jogadas.

Package pattern.observer- Classes FinishGame e StartGame onde está implementado o padrão Observer.

Package model- Package onde estão as classes de interação com os padrões mvp e classes de grande importância para o funcionamento do sistema.

Package pattern.mvp.presenter- Package onde estão reunidos as classes presenter integrantes dos modelos MVP implementados.

Package project- Contém a classe que dá ao arranque ao projeto.

Package tads- Contém as implementações das duas listas usadas.

Package pattern.strategy- Implementação do padrão strategy utilizado na organização da lista adaptável.

Package pattern.mpv.view- Package onde estão reunidos as classes de visualização do projeto incluindo os elementos view dos padrões MVP implementados. (Java Fx).

Interfaces utilizadas

ConfigurationInteration, Interation, SimGameInteration, StatisticsView, View, Edge, Graph, Ranking, Vertex são todas as interfaces utilizadas no sistema, todas utilizadas para definir comportamentos gerais das classes que as implementam.

Algoritmos mais importantes

Nesta área está uma breve explicação dos algoritmos que consideramos serem os mais importantes para o desempenho do sistema.

```java
@Override
public Edge<Connection, Corner> chooseEdge() {
    ArrayList<Edge<Connection, Corner>> list = (ArrayList<Edge<Connection, Corner>>) getAvailableEdges();
    int size = list.size();
    int chosed = size > 2 ? random.nextInt(size - 1) : 0;
    return list.get(chosed);
}
```

Método de escolha de aresta de forma aleatória, usado num jogo de dificuldade fácil.

```java
@Override
public Edge<Connection, Corner> chooseEdge() {
    ArrayList<Edge<Connection, Corner>> list = (ArrayList<Edge<Connection, Corner>>) getAvailableEdges();

    for (Edge<Connection, Corner> e : list) {
        if (isNotIncidentEdge(e)) {
            return e;
        }
    }

    for (Edge<Connection, Corner> e : list) {
        if (!lost(e, getOtherPlayerTurn()) && !lost(e, getPlayerTurn())) {
            return e;
        }
    }

    for (Edge<Connection, Corner> e : list) {
        if (!lost(e, getPlayerTurn())) {
            return e;
        }
    }

    return list.get(0);
}
```

Método de escolha de aresta de forma calculada. O algoritmo escolhe numa primeira ocasião, arestas que não tenham nenhuma incidência com arestas já escolhidas. Quando estas se esgotarem, passa a escolher arestas em que que o oponente não perde assim como ele. No final, se nenhuma destas existir, escolhe uma aresta em que o próprio não perca. Caso todas as arestas disponíveis para escolha o façam perder, a escolha é feita de forma aleatória.

```
private boolean isNotIncidentEdge(Edge<Connection, Corner> edge) {
    HashSet<Edge<Connection, Corner>> set = (HashSet<Edge<Connection, Corner>>) getBoard().getFigure().incidentEdges(edge);
    for (Edge<Connection, Corner> e : set) {
        if (e.element().isSelected()) {
            return false;
        }
    }
    return true;
}
}
```

Para descobrir se uma aresta não tem arestas incidentes já escolhidas, o algoritmo percorre todas as arestas incidentes da aresta dada. Se alguma delas já estiver selecionada é porque existem arestas incidentes usadas. Se nenhuma deles estiver sido selecionada é porque não tem arestas incidentes usadas.

```
@Override
public void update(Observable o, Object arg) {
    if (arg.equals("Start")) {
        Game game = (Game) o;
        String gameType = (game instanceof SimGame) ? " Jogo SimGame( " + game.getPlayers()[0] + " vs " + game.getPlayers()[1]
                + " )" : " jogo ComputerGame( " + game.getPlayers()[0] + " vs " + game.getPlayers()[1] + " ) \n";
        SimpleDateFormat sd = new SimpleDateFormat("dd/MM/yyyy HH:mm");
        Date dataAtual = new Date(System.currentTimeMillis());
        String info="Inicio de Jogo "+ sd.format(dataAtual)+" Tipo: "+gameType;
        try {
            Logger.getInstance().writeToLog(info, Options.GAME);
        } catch (LoggerException ex) {
            //continua mesmo assim
        }
    }
}
```

Observador da classe game que efetua, no início do jogo, o registo de todos os dados e informações a escrever no logger.

```
@Override
public void update(Observable o, Object arg) {
    if (arg.equals("Finish")) {
        Game game = (Game) o;
        String info = "Jogo Acabou - Duração:" + game.getGameTime() + " seguntos\n";
        info += " Vencedor: " + game.getWinnerPlayer() + "\n";
        info += " Perdedor: " + game.getLoserPlayer();
        updateStatistics(game);
        try {
            Logger.getInstance().writeToLog(info, Options.GAME);
        } catch (LoggerException ex) {
            //continua
        }
    }
}

private void updateStatistics(Game game) {
    Player winner = game.getWinnerPlayer();
    Player losser = game.getLoserPlayer();
    TypeOfGames type = getType(game);
    int gameTime=game.getGameTime();
    winner.getIndividualStatistics().incrementNumberGames(type);
    winner.getIndividualStatistics().incrementNumberOfWins(type);
    winner.getIndividualStatistics().incrementTotalOfTime(type, gameTime);
    losser.getIndividualStatistics().incrementNumberOfDefeats(type);
    losser.getIndividualStatistics().incrementNumberGames(type);
    losser.getIndividualStatistics().incrementTotalOfTime(type, gameTime);
    playersManager.getPlayersRegisted().updateStatistics(losser);
    playersManager.getPlayersRegisted().updateStatistics(winner);
}
```

Observador da classe game efetua, no fim do jogo, o registo de todos os dados e informações a escrever no logger e a atualização das estatísticas de ambos os jogadores.

```java
// distribui os vértices pelas respetivas posições de forma a formar um hexágono
private void inicialize() {
    double theta = 2 * Math.PI / 6;
    for (int i = 0; i < 6; i++) {
        figure.insertVertex(new Corner((200 * Math.cos(i * theta) + 300),
                (200 * Math.sin(i * theta) + 275)));
    }

    for (Vertex<Corner> c : figure.vertices()) {
        for (Vertex<Corner> c2 : figure.vertices()) {
            if (c != c2 && !figure.areAdjacent(c, c2)) {
                figure.insertEdge(c, c2, new Connection());
            }
        }
    }
}
```

Método que cria a figura com as respetivas arestas e vértices. É atribuído ao vértice o corner com as posições respetivas dependendo do nº de arestas escolhidas (neste caso 6).

```java
private List<Corner> getTriangle(Edge<Connection, Corner> edge) {
    List<Corner> list = new ArrayList<>();
    Vertex<Corner> v1 = edge.vertices()[0];
    Vertex<Corner> v2 = edge.vertices()[1];
    list.add(v1.element());
    Player p= edge.element().getSeletorUser();
    if(p==null){
        return null;
    }
    for (Edge<Connection, Corner> c : figure.incidentEdges(v1)) {
        Vertex<Corner> v3 = c.vertices()[0];
        Vertex<Corner> v4 = c.vertices()[1];
        if (c != edge && c.element().getSeletorUser() == edge.element().getSeletorUser()) {
            Vertex<Corner> v = (v1 == v3) ? v4 : v3;
            list.add(v.element());
            for (Edge<Connection, Corner> e2 : figure.incidentEdges(v)) {
                Vertex<Corner> v5 = e2.vertices()[0];
                Vertex<Corner> v6 = e2.vertices()[1];
                if (e2 != c && e2.element().getSeletorUser() == edge.element().getSeletorUser() && (v2 == v5 || v2 == v6)) {
                    list.add(v2.element());
                    return list;
                }
            }
            list.remove(v.element());
        }
    }
    return null;
```

Método que devolve a lista dos 3 corners se o edge dado forma um triângulo feito por um jogador. É utilizado para, no fim do jogo acaba mostrar o triângulo formado.

```java
@Override
public int compare(Player p1, Player p2) {
    return p1.getIndividualStatistics().getTotalOfwins()-p2.getIndividualStatistics().getTotalOfwins();
}
```

Método utilizado para comparar os números de vitórias de dois jogadores.

```java
public boolean hasTriangule(Edge<Connection, Corner> edge, Player player) {
    Vertex<Corner> v1 = edge.vertices()[0];
    Vertex<Corner> v2 = edge.vertices()[1];
    for (Edge<Connection, Corner> c : figure.incidentEdges(v1)) {
        Vertex<Corner> v3 = c.vertices()[0];
        Vertex<Corner> v4 = c.vertices()[1];
        if (c != edge && c.element().getSeletorUser() == player) {
            Vertex<Corner> v = (v1 == v3) ? v4 : v3;
            for (Edge<Connection, Corner> e2 : figure.incidentEdges(v)) {
                Vertex<Corner> v5 = e2.vertices()[0];
                Vertex<Corner> v6 = e2.vertices()[1];
                if (e2 != c && e2.element().getSeletorUser() == player && (v2 == v5 || v2 == v6)) {
                    return true;
                }
            }
        }
    }
    return false;
}
```

Método que diz se o jogador dado ao jogar a aresta dada perde o jogo formando um triângulo, serve para saber se já formou o triângulo ou se ao jogar a aresta dada vai formar.

```java
83
84  public void copy(IndividualStatistics statistics) {
        for (Entry<TypeOfGames, Statistic> m : this.statistics.entrySet()) {
86          Statistic s=statistics.getStatistics().get(m.getKey());
87          m.getValue().copyStatistics(s.getNumberOfGames(),
88                  s.getNumberOfWins(),s.getNumberOfDefeats(),s.getTotalOfTime());
89      }
```

Método que serve para copiar o conteúdo de uma estatística para a estatística atual.

```
public int getTotalOfwins() {
    return statistics.get(TypeOfGames.SIMGAME).getNumberOfWins()
            + statistics.get(TypeOfGames.EASYCOMPUTERGAME).getNumberOfWins()
            + statistics.get(TypeOfGames.HARDCOMPUTERGAME).getNumberOfWins();
}

public int getTotalOfGames() {
    return statistics.get(TypeOfGames.SIMGAME).getNumberOfGames()
            + statistics.get(TypeOfGames.EASYCOMPUTERGAME).getNumberOfGames()
            + statistics.get(TypeOfGames.HARDCOMPUTERGAME).getNumberOfGames();
}

public int getTotalOfdefeats() {
    return statistics.get(TypeOfGames.SIMGAME).getNumberOfDefeats()
            + statistics.get(TypeOfGames.EASYCOMPUTERGAME).getNumberOfDefeats()
            + statistics.get(TypeOfGames.HARDCOMPUTERGAME).getNumberOfDefeats();
}

public int getTotalValue() {
    return getTotalOfwins()+getTotalOfGames()+getTotalOfdefeats()+getTotalOftimePlayed();
}


public int getTotalOftimePlayed() {
    return secToMin(statistics.get(TypeOfGames.SIMGAME).getTotalOfTime()
            + statistics.get(TypeOfGames.EASYCOMPUTERGAME).getTotalOfTime()
            + statistics.get(TypeOfGames.HARDCOMPUTERGAME).getTotalOfTime());
}
```

Métodos que retornam o total de todas as estatísticas de vitórias, derrotas, jogos disputados, total de tempo de jogo e o total de estas todas juntas.

Bad Smells encontrados

Nesta fase, fizemos uma análise do código anteriormente implementado de forma a procedermos á sua melhoria através das técnicas de Refactoring aprendidas na cadeira.

Bad Smell	Técnica de refactoring	Classes
Long method	Abstract Method	AccountViewer
		ConfigurationViewer
		GameSeletionViewer
		GameViewer
		SimGameViewer
		LoginViewer
		RegistrationViewer
		VictoryViewer
		StatisticsViewer
		StatisticsPopUp
Duplicate code	Abstract Method	TabsViewer
Message Chain	Hide delegate	Board,computerGameHard

Duplicate Code

```
private void loginBehaviour() {

    login.setOnSelectionChanged(e -> {
        login.setStyle("-fx-background-color: #262626; -fx-opacity: 0.7;");
        login.getGraphic().setStyle("-fx-effect: dropshadow(three-pass-box, #00b3b3, 65, 0.9, 0, 0)");
        game.setStyle("-fx-background-color: grey;");
        game.getGraphic().setStyle("-fx-effect: dropshadow(three-pass-box, white, 65, 0.9, 0, 0)");
        register.getGraphic().setStyle("-fx-effect: dropshadow(three-pass-box, white, 65, 0.9, 0, 0)");
        register.setStyle("-fx-background-color: grey;");
        account.setStyle("-fx-background-color: grey;");
        account.getGraphic().setStyle("-fx-effect: dropshadow(three-pass-box, white, 65, 0.9, 0, 0)");
        statistics.setStyle("-fx-background-color: grey;");
        statistics.getGraphic().setStyle("-fx-effect: dropshadow(three-pass-box, white, 65, 0.9, 0, 0)");
        configuration.setStyle("-fx-background-color: grey;");
        configuration.getGraphic().setStyle("-fx-effect: dropshadow(three-pass-box, white, 65, 0.9, 0, 0)");

    });

    login.getGraphic().setId("login-label");
}

private void registerBehaviour() {
    register.setOnSelectionChanged(e -> {
        register.setStyle("-fx-background-color: #262626; -fx-opacity: 0.7;");
        register.getGraphic().setStyle("-fx-effect: dropshadow(three-pass-box, #00e6b8, 65, 0.9, 0, 0)");
        login.setStyle("-fx-background-color: grey;");
        login.getGraphic().setStyle("-fx-effect: dropshadow(three-pass-box, white, 65, 0.9, 0, 0)");
        game.getGraphic().setStyle("-fx-effect: dropshadow(three-pass-box, white, 65, 0.9, 0, 0)");
        game.setStyle("-fx-background-color: grey;");
        account.setStyle("-fx-background-color: grey;");
        account.getGraphic().setStyle("-fx-effect: dropshadow(three-pass-box, white, 65, 0.9, 0, 0)");
        statistics.setStyle("-fx-background-color: grey;");
        statistics.getGraphic().setStyle("-fx-effect: dropshadow(three-pass-box, white, 65, 0.9, 0, 0)");
        configuration.setStyle("-fx-background-color: grey;");
        configuration.getGraphic().setStyle("-fx-effect: dropshadow(three-pass-box, white, 65, 0.9, 0, 0)");
```

```
    private void style() {
        login.setStyle("-fx-background-color: grey;");
        login.getGraphic().setStyle("-fx-effect: dropshadow(three-pass-box, white, 65, 0.9, 0, 0)");
        game.setStyle("-fx-background-color: grey;");
        game.getGraphic().setStyle("-fx-effect: dropshadow(three-pass-box, white, 65, 0.9, 0, 0)");
        register.getGraphic().setStyle("-fx-effect: dropshadow(three-pass-box, white, 65, 0.9, 0, 0)");
        register.setStyle("-fx-background-color: grey;");
        account.setStyle("-fx-background-color: grey;");
        account.getGraphic().setStyle("-fx-effect: dropshadow(three-pass-box, white, 65, 0.9, 0, 0)");
        statistics.setStyle("-fx-background-color: grey;");
        statistics.getGraphic().setStyle("-fx-effect: dropshadow(three-pass-box, white, 65, 0.9, 0, 0)");
        configuration.setStyle("-fx-background-color: grey;");
        configuration.getGraphic().setStyle("-fx-effect: dropshadow(three-pass-box, white, 65, 0.9, 0, 0)");
    }

    private void loginBehaviour() {
        login.setOnSelectionChanged(e -> {
            style();
            login.setStyle("-fx-background-color: #262626; -fx-opacity: 0.7;");
            login.getGraphic().setStyle("-fx-effect: dropshadow(three-pass-box, #00b3b3, 65, 0.9, 0, 0)");
        });

        login.getGraphic().setId("login-label");
    }

    private void registerBehaviour() {
        register.setOnSelectionChanged(e -> {
            style();
            register.setStyle("-fx-background-color: #262626; -fx-opacity: 0.7;");
            register.getGraphic().setStyle("-fx-effect: dropshadow(three-pass-box, #00e6b8, 65, 0.9, 0, 0)");
        });
        register.getGraphic().setId("register-label");
    }
```

Message Chain

```
private boolean isNotIncidentEdge(Edge<Connection, Corner> edge) {
    HashSet<Edge<Connection, Corner>> set = (HashSet<Edge<Connection, Corner>>) getBoard().getFigure().incidentEdges(edge);
    for (Edge<Connection, Corner> e : set) {
        if (e.element().isSelected()) {
            return false;
        }
    }
    return true;
}
```

```
private boolean isNotIncidentEdge(Edge<Connection, Corner> edge) {
    HashSet<Edge<Connection, Corner>> set = (HashSet<Edge<Connection, Corner>>) getBoard().incidentEdges(edge);
    for (Edge<Connection, Corner> e : set) {
        if (e.element().isSelected()) {
            return false;
        }
```

Long Method

```
private void setupLayout() {
    ivExit.setFitHeight(33);
    ivExit.setFitWidth(33);
    btnExit.setTranslateX(463);
    btnExit.setTranslateY(502);

    logout.setFitHeight(30);
    logout.setFitWidth(30);
    logout.setTranslateX(38);
    logout.setTranslateY(486);
    logout.setId("logout-view");

    ivLogout1.setFitHeight(50);
    ivLogout1.setFitWidth(50);
    btnLogoutUser1.setTranslateX(100);
    btnLogoutUser1.setTranslateY(490);
    btnLogoutUser1.setVisible(false);
    btnLogoutUser1.setId("logout-button");

    ivLogout2.setFitHeight(51);
    ivLogout2.setFitWidth(50);
    btnLogoutUser2.setTranslateX(144);
    btnLogoutUser2.setTranslateY(490);
    btnLogoutUser2.setVisible(false);
    btnLogoutUser2.setId("logout-button");

    logoutLabel.setTranslateX(-196);
    logoutLabel.setTranslateY(96);
    logoutLabel.setId("credential-label");
```

Continua…

```
private void setupExit() {
    ivExit = new ImageView("/Images/exit.png");
    ivExit.setFitHeight(33);
    ivExit.setFitWidth(33);

    btnExit = new Button("", ivExit);
    btnExit.setTranslateX(463);
    btnExit.setTranslateY(502);

    exitLabel = new Label("Sair");
    exitLabel.setTranslateX(210);
    exitLabel.setTranslateY(52);
    exitLabel.setId("credential-label");
}

private void setupLayout() {
    setupBtnEnter();
    setupExit();
    setupLine();
    setupLogoutButton();
    setupLogoutUser1();
    setupLogoutUser2();
    setupVisiblePassword();
    setupPassword();
    setupShow();
    setupTranslations();
    setupUserIcon();
    setupUsername();
    setupWellcomeLabel();
    setupPasswordIcon();
    getChildren().addAll(btnExit, logout, btnLogoutUser1, btnLogoutUser2, logoutLabel, exitLabel);
}
```

Para além dos Bad smells e correções mostradas, existiram outras implementações no código. Nomeadamente algumas restruturações de forma a facilitar a leitura do código.

Conclusões

A elaboração deste projeto permitiu-nos desenvolver capacidades a diversos níveis. Desde o desenvolvimento de algoritmos em Java, até á implementação de estruturas como os grafos, padrões de software e aplicação de técnicas de melhoramento de código. Em suma, componentes muito importantes para o nosso futuro na área. Em relação ás maiores dificuldades sentidas, consideramos que passaram pela aplicação dos padrões de software visto ter sido o nosso primeiro contacto com a matéria.