Relatório

Trabalho – Snake & Ladders

Licenciatura de Engenharia da Computação Gráfica e Multimédia 2º Ano

Unidade Curricular: Laboratório de programação

IPVC-ESTG

16 de junho 2024

Docente: Luís Romero
Discente: Marco Fernandes; nº30566
Rui Fernandes; nº20092

INTRODUÇÃO

O presente relatório introduz-se no segundo ano do plano curricular da licenciatura de Engenharia da Computação Gráfica e Multimédia, da Escola Superior de Tecnologia e Gestão – IPVC, no âmbito da unidade curricular de Laboratório de programação.

A turma dividiu-se em grupos de dois e três elementos e escolheram um jogo dado pelo docente da unidade curricular ou poderiam propor um novo que fosse aprovado pelo mesmo.

Como tal, o presente grupo escolheu um jogo bastante conhecido chamado “Snake and Ladders”.

O objetivo principal deste trabalho é implementar o jogo em JavaFX utilizando o IntelliJ com o auxílio do SceneBuilder e também um servidor que permita a jogabilidade com dois jogadores.

Figura 1 - Logo do IntellIJ e Scene Builder

OBJETIVOS E REGRAS DO JOGO

Objetivos de jogo

O objetivo do jogo é ser o primeiro jogador a chegar à última casa do tabuleiro, que é a casa de número 100 ou uma estrela. Durante o percurso os jogadores poderão ter a probabilidade de obter uma ajuda ou um azar, deixando assim o jogo mais desafiante.

Regras do Jogo

1. **Preparação**
    - **a.** Cada jogador escolhe uma peça.
    - **b.** Os jogadores decidem a ordem de jogada (neste caso vai se jogar o dado e quem tirar o número 6 começa).
2. **Movimentação**
    - **a.** O jogador "move" seu peão o número de casas correspondente ao valor tirado no dado.
3. **Escadas**
    - **a.** Se um jogador parar na base de uma escada, ele sobe até o topo da escada. As escadas ajudam o jogador a avançar no tabuleiro.
4. **Cobras**
    - **a.** Se um jogador parar na cabeça de uma cobra, ele desce até a cauda da cobra. As cobras fazem o jogador retroceder no tabuleiro.
5. **Interações**
    - **a.** Se um jogador cair na mesma casa de outro jogador, não há penalidade ou ação especial, ou seja, ambos os jogadores permanecem nas mesmas casas até a próxima jogada.
    - **b.** Para ganhar, é preciso tirar o número no dado preciso para a casa final (exemplo: se o jogador tiver na casa 99, terá que tirar 1 no dado para ganhar, se tirar mais, terá que voltar a tentar).

MANUAL DE UTILIZAÇÃO

Após iniciar o jogo, um ecrã com três botões será exibido. O primeiro botão, "Jogar", redireciona para a página de inserção dos dados. O segundo, "Regras", exibe as instruções do jogo. O terceiro botão, "Sair", permite ao jogador fechar o jogo.

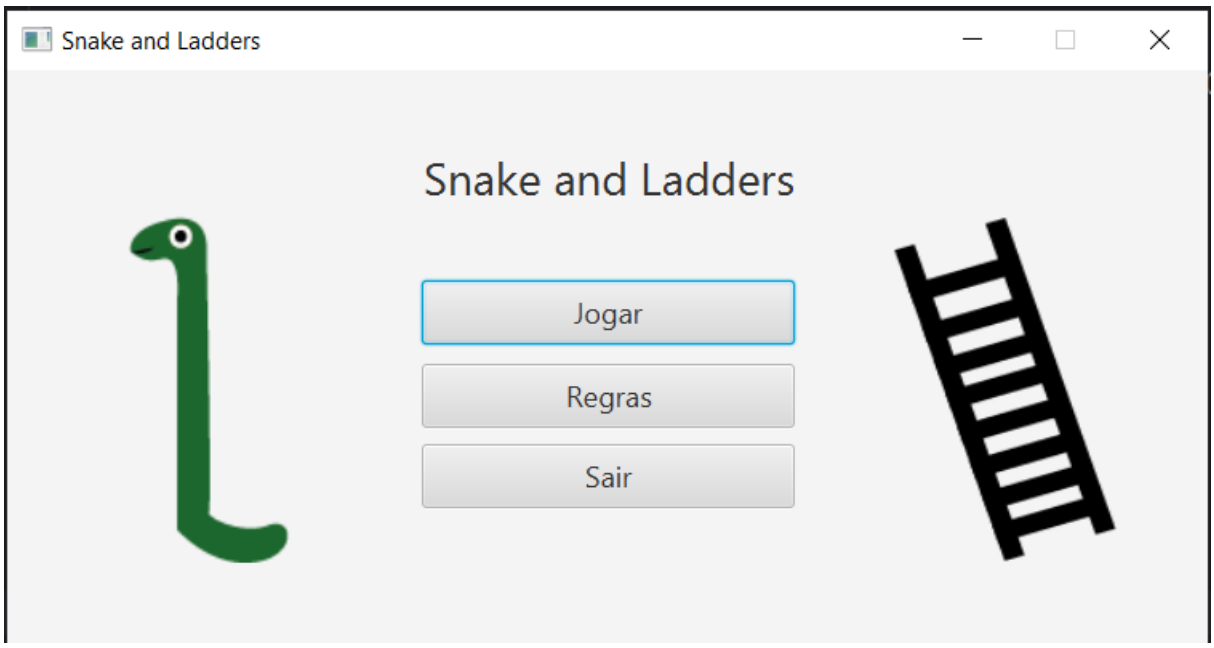

Figura 2 - Interface inicial do jogo

A página das regras encontra-se demonstrada na figura abaixo.

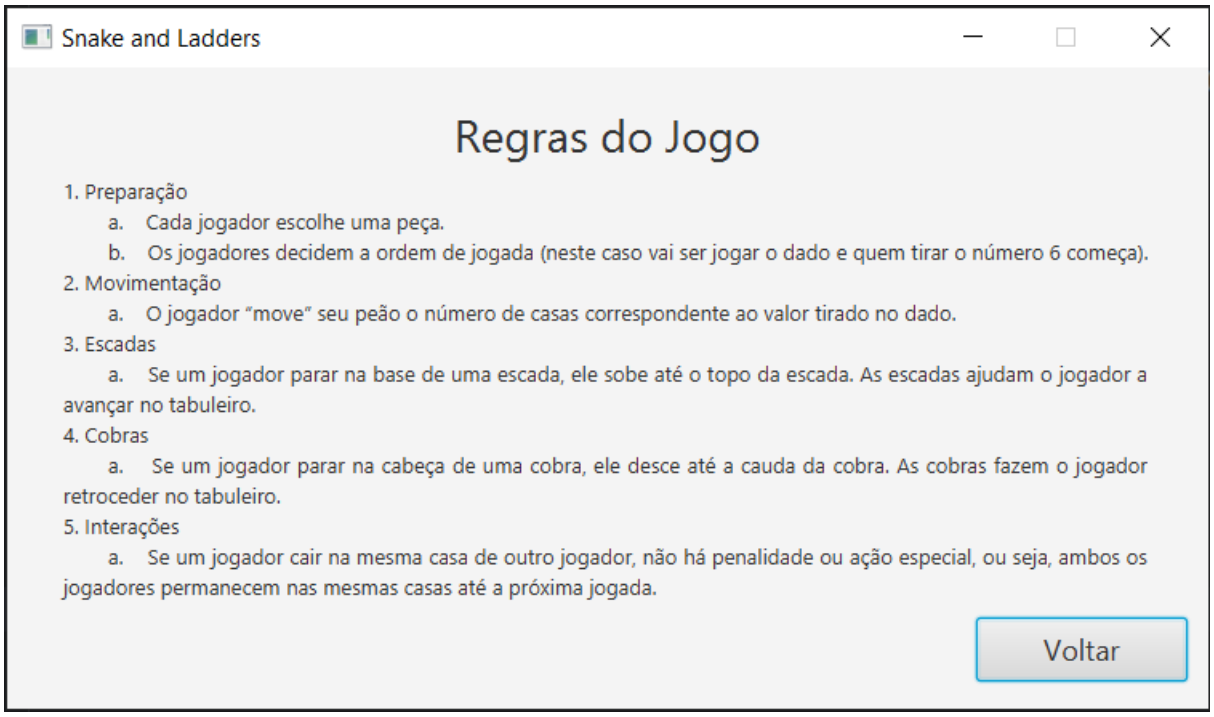

Figura 3 - Regras

Quando o jogador clica em “jogar” será mostrado uma página ao qual o jogador escolherá a cor da sua peça e três inputs para o nome, o IP e a porta respetivamente. Caso não insira os campos todos aparecerá uma mensagem de erro como se pode observar na figura abaixo.

Figura 4 - Interface da inserção dos dados

Assim que for conectado ao servidor, a interface do jogo é constituída por o tabuleiro, uma caixa de texto que “anima” de acordo com os avisos que será necessário transmitir ao jogador, como por exemplo ao iniciar o jogo ou jogadas, ou seja se é a vez do oponente ou a vez do jogador, um dado e dois botões, um para rodar e outro para sair/desistir.

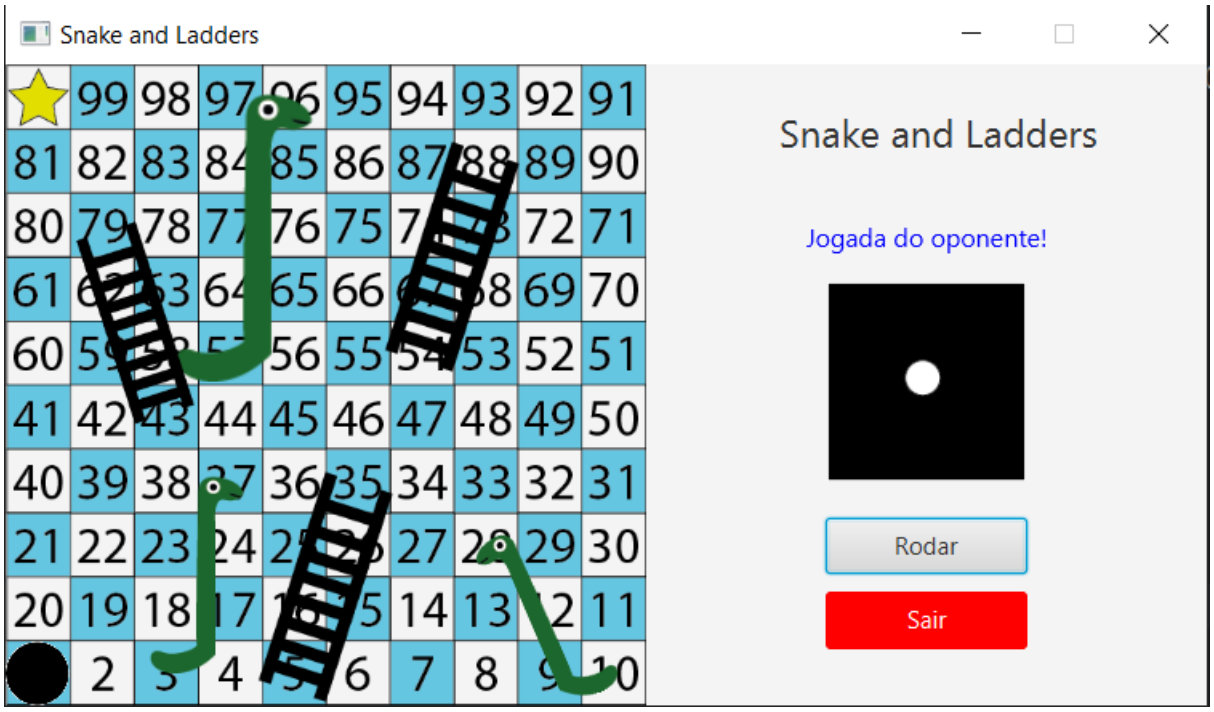

Figura 5 - Interface do jogo

Caso o jogador queira desistir, aparecera uma mensagem de aviso/confirmação para o efeito, como se observa na figura seguinte.

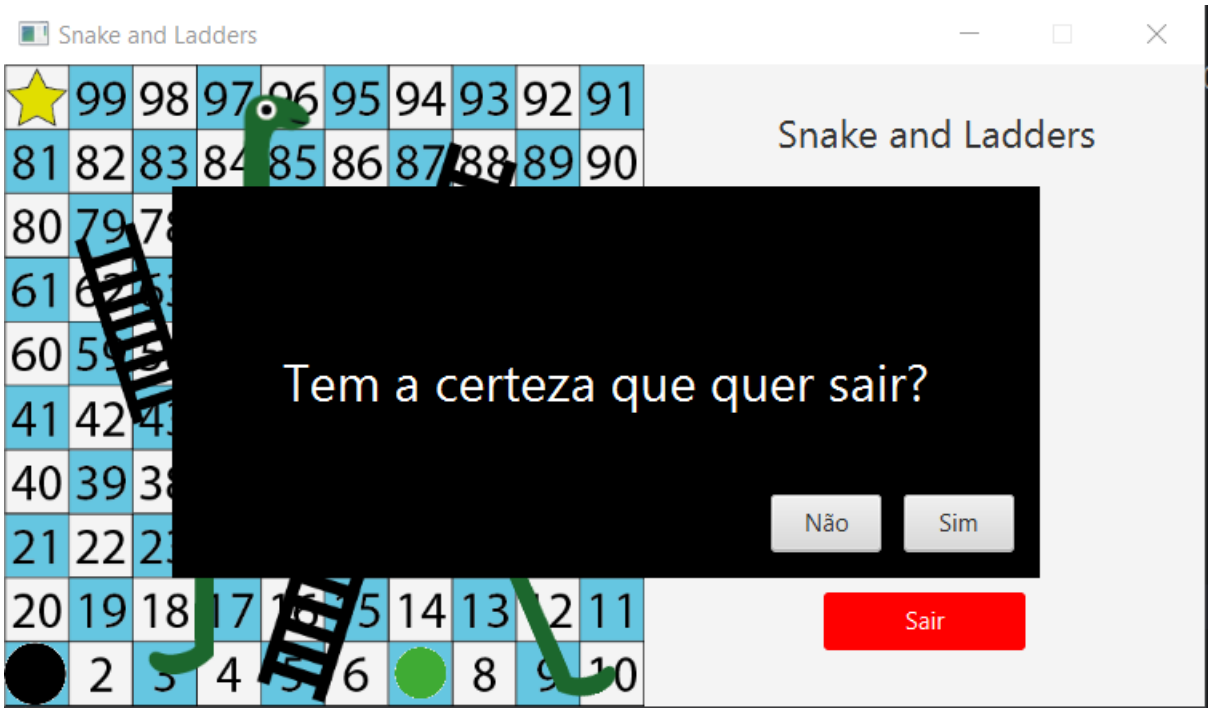

Figura 6 - Aviso de confirmação de desistência

Caso seja o oponente a desistir, aparecerá o seguinte aviso.

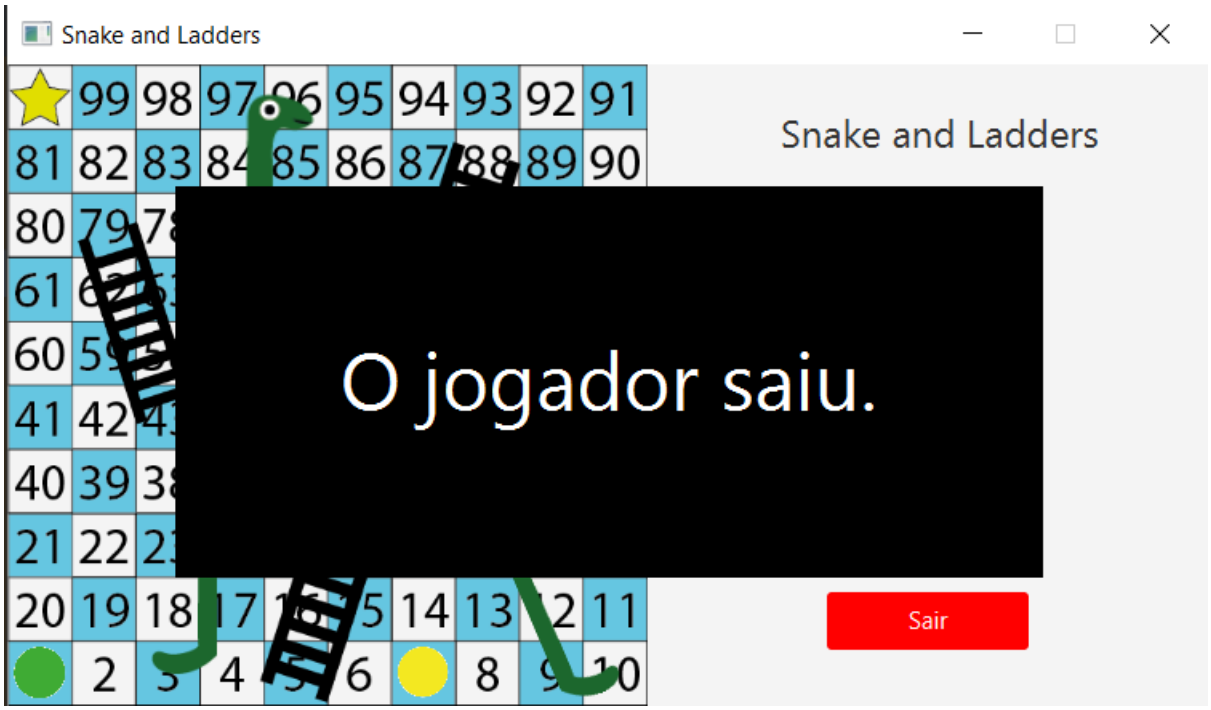

Figura 7 - Aviso da desistência do oponente

Se um jogador "cair" numa cobra, ele descerá para a casa correspondente situada na cauda, se for num escadote ele subirá até ao topo da mesma. O primeiro jogador a chegar a estrela ou casa número 100 (número certeiro no dado ou não sai do sítio) será o vencedor e aparecerá uma mensagem de aviso. No lado do jogador que perdeu aparecerá também uma mensagem de aviso.

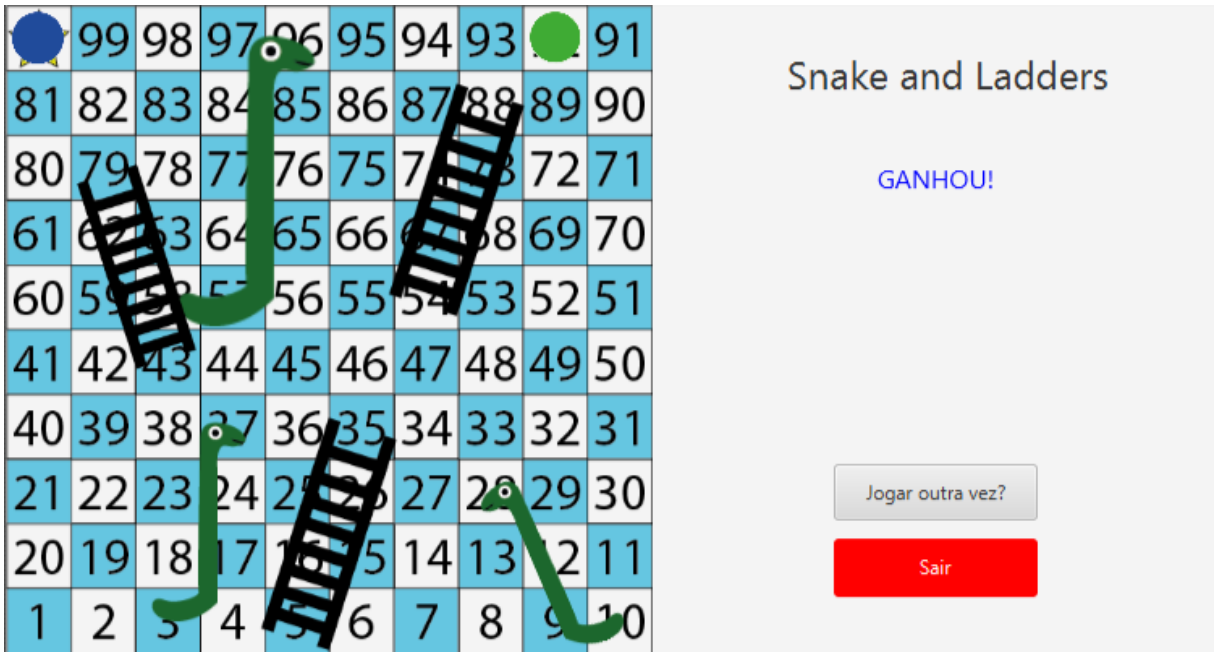

Figura 8 - Vencedor

No lado do jogador que perdeu aparecerá também uma mensagem de aviso.

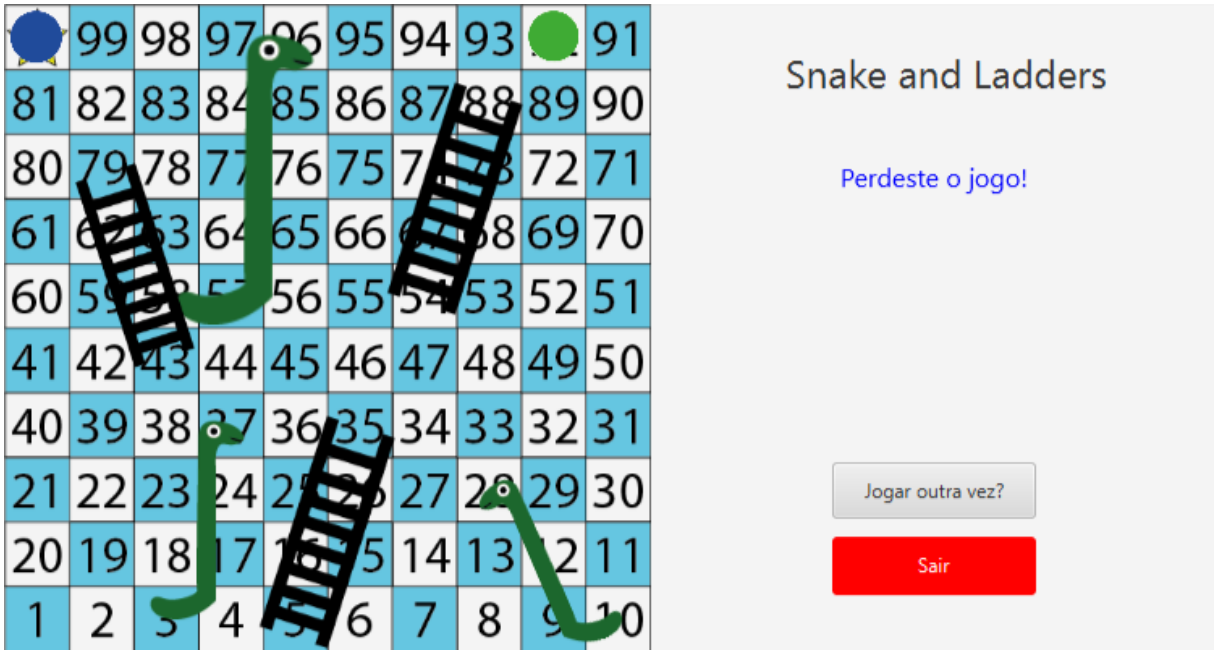

Figura 9 - Derrotado

FUNCIONALIDADES

Estrutura de classes

Para a implementação deste trabalho foi implementado a seguinte estrutura de dados composta por seis classes mais o ficheiro FXML e também uma pasta com as imagens desenhadas em Illustrator, como se pode ver na seguinte figura.

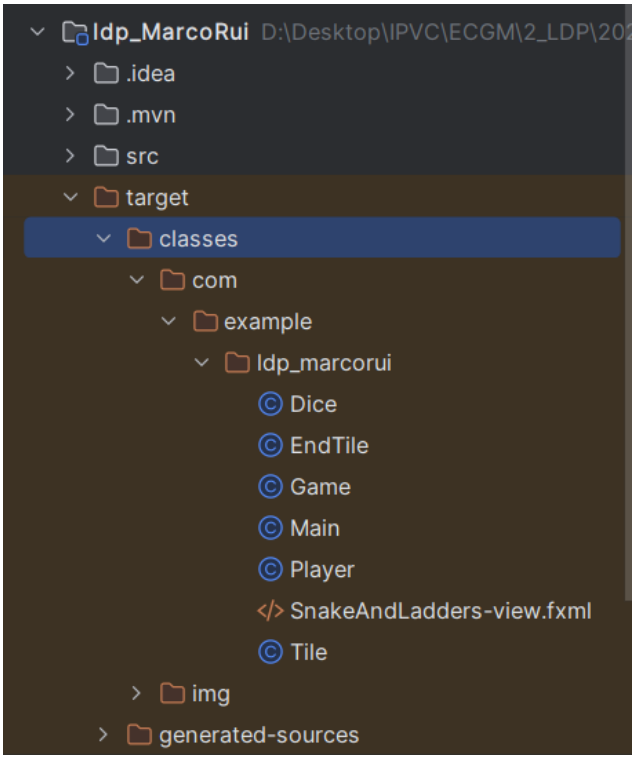

Figura 10 - Estrutura de classes

Classe Dice

A classe Dice representa o dado utilizado no jogo. Praticamente esta classe carrega num array, as imagens dos seis lados do dado e simula um lançamento do mesmo, retornando um valor.

```
public static int roll() {
    return (int)(Math.random() * 6.0 + 1.0);
}
```

Figura 11 - Função que simula o lançamento do dado

Classe EndTile

A classe EndTile representa uma extremidade seja a cabeça da cobra ou o topo da escada. Praticamente cria uma nova extremidade ao Tile especificado.

```
public class EndTile {  no usages
    Tile endTile;

    public EndTile(Tile endTile) {
        this.endTile = endTile;
    }
}
```

Figura 12 - Classe EndTile

Classe Game

Esta classe é responsável por controlar a logica do jogo. Ela inclui métodos para configurar a interface, manipular eventos de entrada do jogador, gerir a comunicação de rede e implementar a lógica do jogo. Pode-se afirmar que esta classe é a que contém a grande maioria do código implementado.

```
EndTile cobra1 = new EndTile((Tile)this.board.get(2));
((Tile)this.board.get(36)).setSnake(cobra1);
EndTile cobra2 = new EndTile((Tile)this.board.get(9));
((Tile)this.board.get(27)).setSnake(cobra2);
EndTile cobra3 = new EndTile((Tile)this.board.get(57));
((Tile)this.board.get(95)).setSnake(cobra3);
EndTile escada1 = new EndTile((Tile)this.board.get(34));
((Tile)this.board.get(4)).setLadder(escada1);
EndTile escada2 = new EndTile((Tile)this.board.get(87));
((Tile)this.board.get(53)).setLadder(escada2);
EndTile escada3 = new EndTile((Tile)this.board.get(78));
((Tile)this.board.get(42)).setLadder(escada3);
```

Figura 13 - Snake and Ladders

Nesta classe também se faz o tratamento de dados da conexão ao servidor como se verifica na figura 14.

```
private boolean connect() {
    try {
        this.socket = new Socket(this.ip, this.port);
        this.dos = new DataOutputStream(this.socket.getOutputStream());
        this.dis = new DataInputStream(this.socket.getInputStream());
        this.dos.writeUTF(this.selectedColor);
        String serverColor = this.dis.readUTF();
        this.imgPlayerOne = new Image( url: "/img/" + this.selectedColor + ".png");
        this.playerOneColor.setImage(this.imgPlayerOne);
        this.playerOneColor.setVisible(true);
        this.imgPlayerTwo = new Image( url: "/img/" + serverColor + ".png");
        this.playerTwoColor.setImage(this.imgPlayerTwo);
        this.playerTwoColor.setVisible(true);
        this.isAccepted = true;
        this.isPlayerTwoTurn = true;
    } catch (IOException var2) {
        System.out.println("Não foi possível connectar ao endereço: " + this.ip + ": " + this.port + ".");
        return false;
    }

    System.out.println("Conexão ao servidor feita com sucesso!");
    return true;
}
```

Figura 14 - Conexão ao servidor

Na classe Game é também implementado o serviço de informação ao jogador sobre o que se está a passar no jogo. Na figura seguinte, por exemplo, esta demonstrado e implementado as mensagens de aguardar pelo o jogador oponente e de quem é a vez de jogar.

```
if (!this.isPlayerTwoTurn && !this.isAccepted) {
    this.textBeforeValue.setText("À espera do oponente...");
    this.textBeforeValue.setLayoutX(491.0);
    this.playerOne = new Player(this.nameInput, this.playerOneColor);
    this.imgPlayerOne = new Image( url: "/img/" + this.selectedColor + ".png");
    this.playerOneColor.setImage(this.imgPlayerOne);
    this.playerOneColor.setVisible(true);
    this.playerOne.setTilePlayer((Tile)this.board.get(0));
    this.playerOne.getColor().setLayoutX((double)((Tile)this.board.get(0)).getX());
    this.playerOne.getColor().setLayoutY((double)((Tile)this.board.get(0)).getY());
} else {
    if (this.isPlayerTwoTurn) {
        this.textBeforeValue.setLayoutX(500.0);
        this.textBeforeValue.setText("Jogada do oponente!");
    } else {
        this.textBeforeValue.setLayoutX(504.0);
        this.textBeforeValue.setText("É a tua vez de jogar!");
```

Figura 15 - Informações para jogador

Implementou-se também o método run que é responsável por executar um loop continuo que controla a movimentação do oponente. Ele também faz requisições ao servidor baseado no estado da jogada e na sua aceitação assim como uma breve pausa em cada iteração.

```
public void run() {
    while(true) {
        if (!this.playerLeft) {
            this.opponentMove();
        }

        if (!this.isPlayerTwoTurn && !this.isAccepted) {
            this.serverRequest();
        }

        try {
            Thread.sleep( millis: 10L);
        } catch (InterruptedException var2) {
            Thread.currentThread().interrupt();
            return;
        }
    }
}
```

Figura 16 - Método Run

Classe Main

A classe main é a classe principal que inicia o jogo. Carrega também as imagens do dado.

```
public class Main extends Application {
    public Main() {
    }

    public void start(Stage stage) throws IOException {
        Parent root = (Parent)FXMLLoader.load(this.getClass().getResource( name: "SnakeAndLadders-view.fxml"));
        stage.setTitle("Snake and Ladders");
        stage.setScene(new Scene(root));
        stage.setResizable(false);
        stage.show();
    }

    public static void main(String[] args) {
        Dice.loadImage();
        launch(args);
    }
}
```

Figura 17 - Classe Main

Classe Player

A classe Player é a classe que representa um jogador. É nesta classe que se define o nome do jogador assim como a Tile onde o jogador se encontra e também a cor da sua peça.

```
public void setName(String name) { this.name = name; }

public String getName() { return this.name; }

public ImageView getColor() { return this.color; }

public void setTilePlayer(Tile tilePlayer) { this.tilePlayer = tilePlayer; }

public Tile getTilePlayer() { return this.tilePlayer; }

public boolean getCanAdvance() { return this.canAdvance; }

public void setCanAdvance(boolean canAdvance) { this.canAdvance = canAdvance; }
```

Figura 18 - Gets e Sets da classe Player

Classe Tile

A classe Tile representa um único tile no jogo. Criando cada um com um id (número da "casa" que aparecerá ao jogador) e as coordenadas especificas. É aqui que se obtém as coordenadas X e Y associada ao tile assim como as cobras e escadas.

```
private int id;
private int x;
private int y;
private EndTile snake = null;
private EndTile ladder = null;
```

Figura 19 - Variáveis da Classe Tile

Javadoc

Javadoc é um gerador de documentação que serve para documentar os programas em Java, a partir do código-fonte. O resultado é expresso em HTML. É constituído, basicamente, por algumas marcações muitos simples inseridas nos comentários do programa.

Como tal, o Javadoc foi falado em aula e serve como requisito na proposta do trabalho presente. Na figura seguinte vimos a documentação em HTML gerado por o IntellIJ. Dentro de cada classe tem toda a explicação dos métodos implementados.

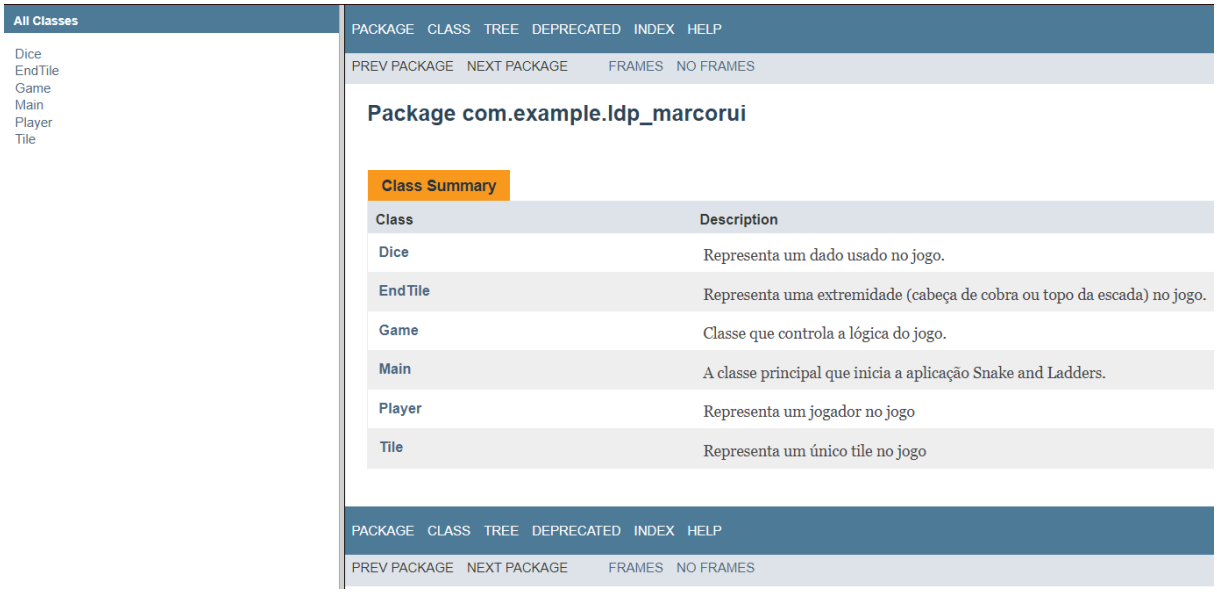

Figura 20 - JavaDoc gerado

Diagrama de classes

O diagrama de classes que representa o trabalho implementado será apresentado na figura seguinte.

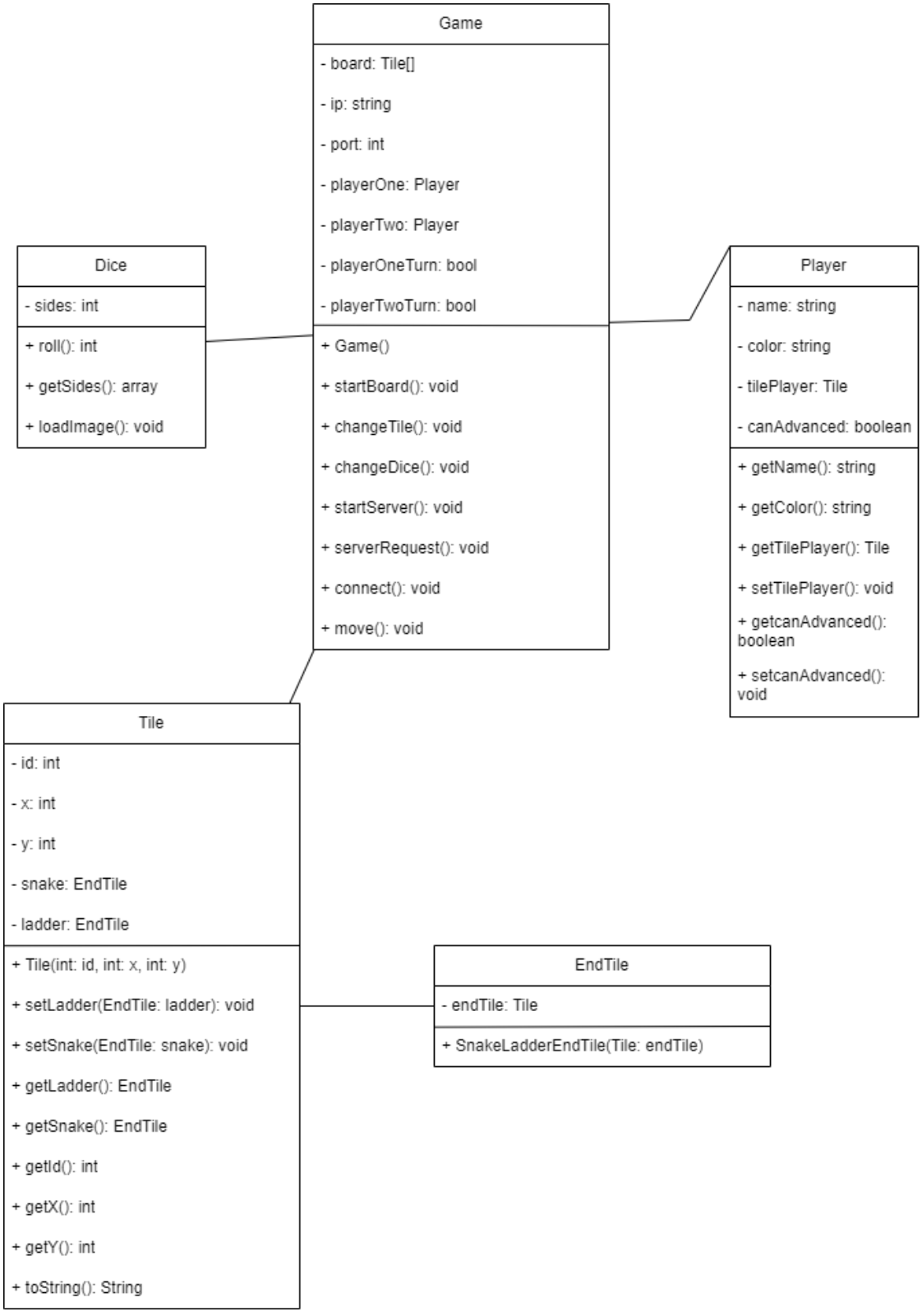

Figura 21 - Diagrama de classes

GITHUB

O GitHub é uma plataforma de hospedagem de código-fonte (baseado numa Cloud) que utiliza o Git para controlo de versões.

Foi fundado em 2008, com o intuito de permitir aos programadores realizarem projetos a distância uma vez que permite a edição de ficheiros e o armazenamento de um registo detalhado das alterações.

Para a implementação deste trabalho foi bastante utilizado para gestor de versões desde das apresentações iniciais, ao design/Mockup e o próprio trabalho final.

Para tal, para alem do "main" principal, criou-se mais duas branches para cada elemento do grupo identificada por "main_Rui" e "main_Marco" como se pode verificar na figura abaixo.

Ao lado verifica-se o gráfico de histórico da evolução de trabalho por parte dos elementos do grupo.

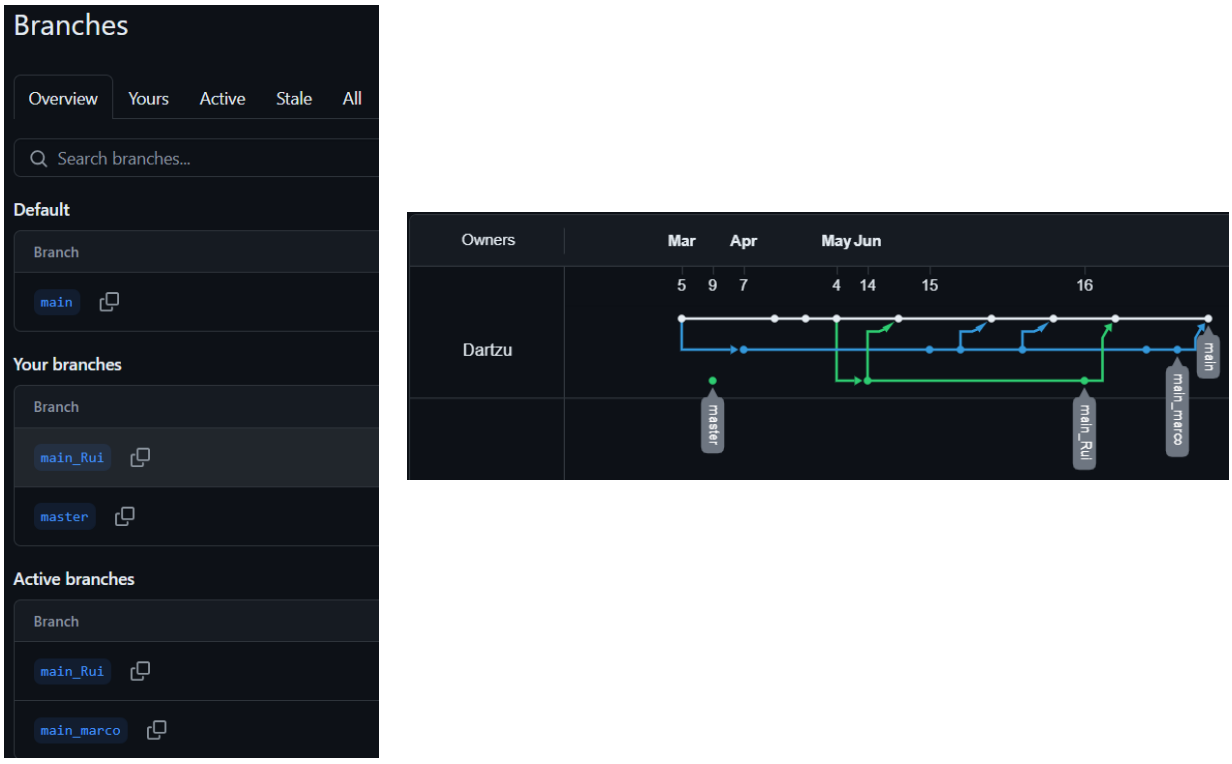

Figura 22 - Branches e historico de evolução

CONCLUSÃO

A implementação do jogo "Snake and Ladders" em JavaFX, com auxílio do Scene Builder e um servidor desenvolvido internamente, foi um projeto enriquecedor e desafiador. Serviu para adquirir competências que sem dúvidas serão necessárias e essenciais no mercado do trabalho.

Surgiram algumas dificuldades aos quais foram ultrapassadas com a entreajuda do grupo e os apontamentos disponibilizados no Moodle.

O projeto resultou num jogo funcional e interativo onde os jogadores podem jogar em tempo real e desfrutar de uma interface gráfica amigável. A implementação bem-sucedida deste jogo não só demonstra o domínio das tecnologias utilizadas, mas também a capacidade de enfrentar e resolver problemas complexos de desenvolvimento de software.

ANEXO

Link para o GITHUB: https://github.com/Dartzu/SnakeAndLeddersLDP