Relatório Meta 2

Programação Avançada

Instituto Superior de Engenharia Informática

Turma Prática 3

João Carvalho

2019131769

1 de maio de 2023

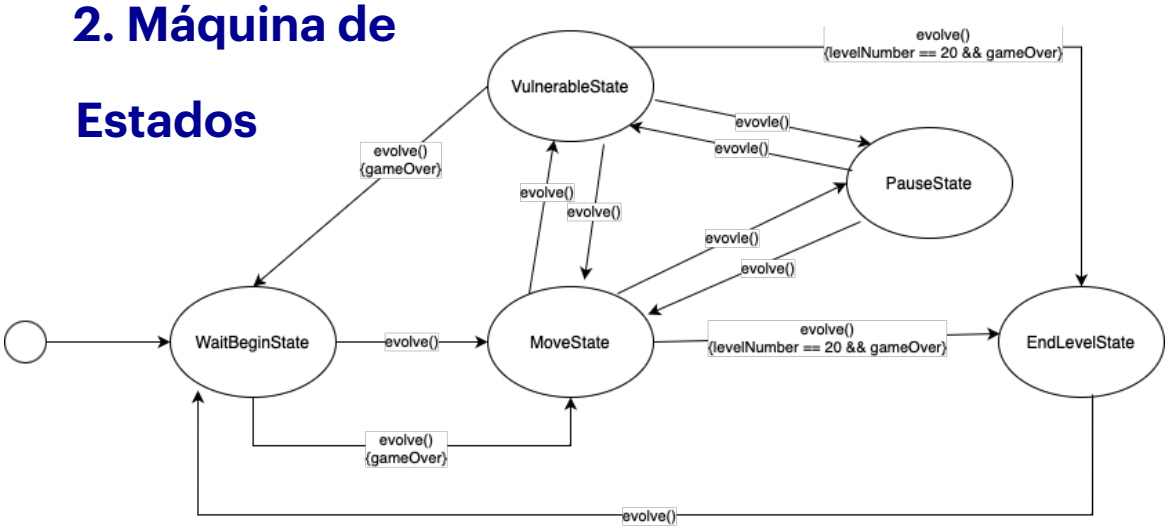
1. Descrição

O objetivo deste trabalho foi a implementação de um jogo tipo PacMan.

Foi implementada um máquina de estados para gerir os estados do jogo. O jogo também possui elementos divididos em packages, sendo estes correspondentes aos intervenientes no jogo como o PacMan e os Fantasmas, as Frutas, a Comida entre outros.

Nesta meta foi implementada uma interface em JavaFX para o utilizador poder usufruir da aplicação e interagir com a mesma.

2. Máquina de Estados



Esta máquina de estados é constituída por 5 estados:

- WaitBeginState: estado em que o jogo está a espera que seja pressionada uma tecla para começar.
- MoveState: estado em que o jogo decorre e onde os “Mobs” se movimentam.
- VulnerableState: estado do jogo iniciado após o PacMan ingerir uma bola com poderes, tornando os Fantasmas vulneráveis e fazendo com que o PacMan os possa comer.
- PauseState: estado do jogo que é apresentado quando o jogador carrega na tecla Escape.
- EndLevelState: estado de jogo que representa o final do jogo após o jogador ter perdido ou ter ganho.

3. Classes

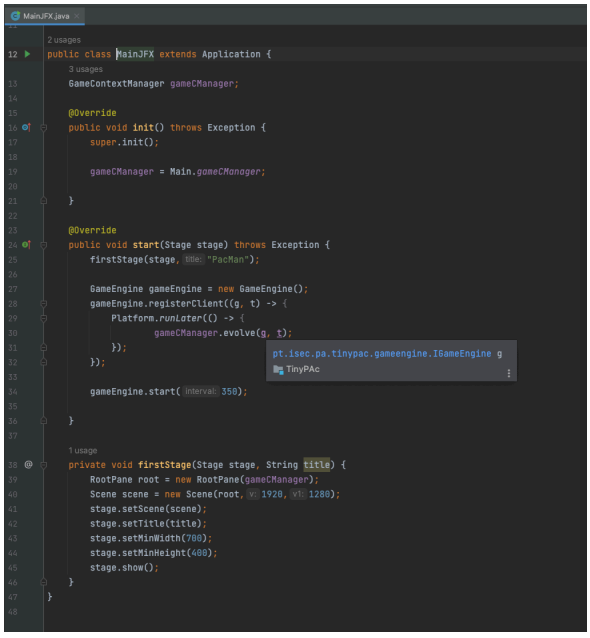
```java
package pt.isec.pa.tinypac;

import pt.isec.pa.tinypac.model.GameContextManager;
import pt.isec.pa.tinypac.ui.gui.MainJFX;

import javafx.application.Application;

2 usages
public class Main {
    public static GameContextManager gameCManager;
    static{
        gameCManager = new GameContextManager();
    }
    no usages
    public static void main(String[] args) { Application.launch(MainJFX.class,args); }
}
```

Main:

- Classe onde são inicializados o levelManager, o context, a GameEngine e as UI´s.

MainJFX:

- Classe onde é iniciado o stage e o RootPane.

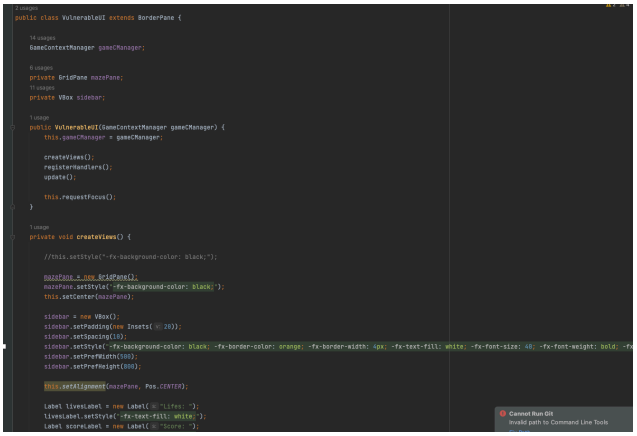


```java
2 usages
public class MainJFX extends Application {
    3 usages
    GameContextManager gameCManager;

    @Override
    public void init() throws Exception {
        super.init();

        gameCManager = Main.gameCManager;

    }

    @Override
    public void start(Stage stage) throws Exception {
        firstStage(stage, title: "PacMan");

        GameEngine gameEngine = new GameEngine();
        gameEngine.registerClient((g, t) -> {
            Platform.runLater(() -> {
                    gameCManager.evolve(g, t);
            });
        });

        gameEngine.start( interval: 350);

    }

    1 usage
    private void firstStage(Stage stage, String title) {
        RootPane root = new RootPane(gameCManager);
        Scene scene = new Scene(root, v: 1920, v1: 1280);
        stage.setScene(scene);
        stage.setTitle(title);
        stage.setMinWidth(700);
        stage.setMinHeight(400);
        stage.show();
    }
}
```

```java
2 usages
public class WaitBeginUI extends BorderPane {
    5 usages
    GameContextManager gameCManager;

    4 usages
    Text waitText;

    1 usage
    public WaitBeginUI(GameContextManager gameCManager) {
        this.gameCManager = gameCManager;

        createViews();
        registerHandlers();
        update();

        this.requestFocus();
    }

    1 usage
    private void createViews() {
        this.setStyle("-fx-background-color: black;");

        waitText = new Text( s: "Pressione uma tecla para começar");
        waitText.setFont(Font.font( s: "Verdana", FontWeight.BOLD, v: 24));
        waitText.setFill(Color.WHITE);

        VBox vbox = new VBox(waitText);
        vbox.setAlignment(Pos.CENTER);
        this.setCenter(vbox);
    }

    1 usage
    private void registerHandlers() {
        gameCManager.addPropertyChangeListener(evt -> { update(); });
        setOnKeyPressed(event -> {
            gameCManager.evolve();
        });
    }

    2 usages
```

WaitBeginUI:

-Classe do apresenta no ecrã o que corresponde ao waitBeginState.

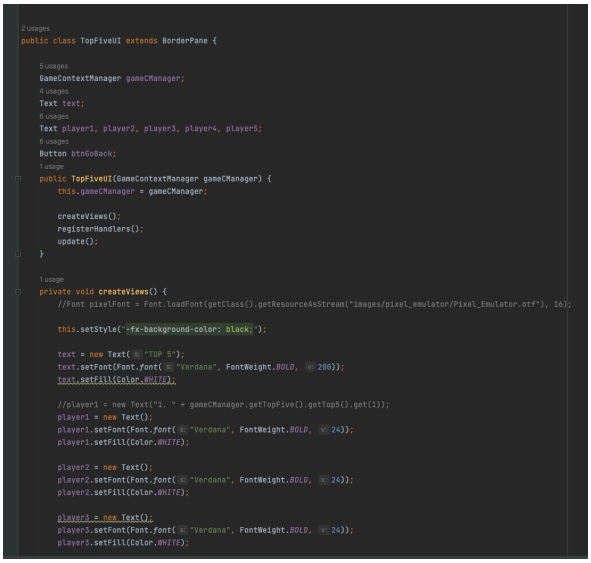


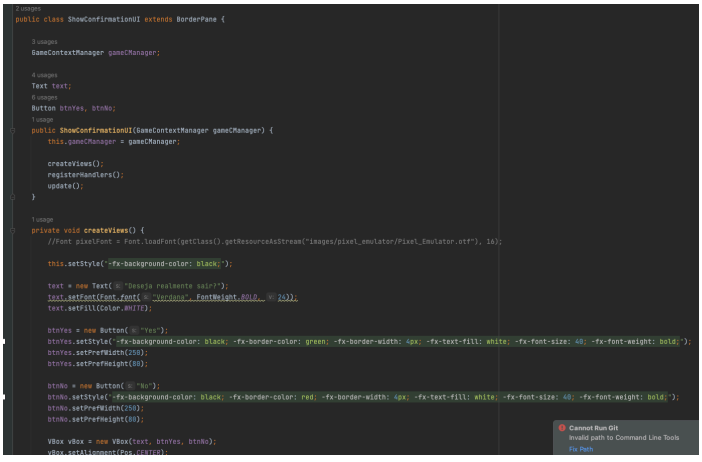
VulnerableUI:

- Classe que apresenta no ecrã o que corresponde ao VulnerableState

TOP5UI:

- Classe que apresenta no ecrã o menu do Top5.





ShowConfirmationUI:

-Classe que mostra uma mensagem de confirmação se quer sair.

```java
public class GameContext {

    3 usages
    Game game;

    6 usages
    IMobsState gameState;

    sudo-john-blossom
    public GameContext(){
        game = new Game(/*"Level101.txt"*/ 1);
        this.gameState = new WaitBeginState( context: this, game);
    }

    1 usage  sudo-john-blossom
    public EMobsState getState(){return gameState.getState();} //foi dado override no MenuState para ele poder ir buscar o state

    1 usage  sudo-john-blossom
    void changeState(IMobsState newState){this.gameState = newState;}

    sudo-john-blossom
    public boolean move(){return gameState.move();}

    1 usage  sudo-john-blossom
    public boolean vulnerable(){return gameState.vulnerable();}

    3 usages  sudo-john-blossom
    public boolean endLevel(){return gameState.endLevel();}

    //getData
    no usages  sudo-john-blossom
    public Game getGame(){return game;}
}
```

GameContext:

- Classe responsável por cuidar do jogo e dos estados do jogo.

GameContextManager:

- Classe que serve como proxy entre o gameContext e o resto do programa.

```java
public class GameContextManager implements IGameEngineEvolve {

    2 usages
    private static final String SAVE_FILE ="files/save.dat";
    2 usages
    private static final String TOP5_FILE ="files/topFive.dat";
    23 usages
    private GameContext fsm;
    12 usages
    PropertyChangeSupport pcs;

    public GameContextManager() { pcs = new PropertyChangeSupport( sourceBean: this); }

    public GameContext getFsm() { return fsm; }
    1 usage
    public void setFsmNull(){fsm = null;}

    public void addPropertyChangeListener(PropertyChangeListener listener){pcs.addPropertyChangeListener(listener);}

    public void start(){
        fsm = new GameContext();
        pcs.firePropertyChange( propertyName: null,  oldValue: null,  newValue: null);
    }

    public EMobsState getState(){return fsm.getState();}

    public boolean evolve(){
        //if(fsm == null) return false;

        var ret = fsm.evolve();
        pcs.firePropertyChange( propertyName: null,  oldValue: null,  newValue: null);
        return ret;
    }//evolve de mudança de estado

    public boolean pause(){
        var ret = fsm.pause();
        pcs.firePropertyChange( propertyName: null,  oldValue: null,  newValue: null);
```

```java
public class EndLevelState extends MobsStateAdapter {

    1 usage  sudo-john-blossom
    public EndLevelState(GameContext context, Game game){
        super(context, game);

        //SETTERS
    }

    1 usage  sudo-john-blossom
    @Override
    public boolean endLevel(){
        //aqui vai mudar o level
        //game.setLevel(new Level(game.getLevel().getLevelNumber() + 1)); // demonstraçao que vai para o nivel seguinte
        changeState(EMobsState.WAIT_BEGIN);
        return true;
    }

    1 usage  sudo-john-blossom
    @Override
    public EMobsState getState(){return EMobsState.END_LEVEL;}
}
```

States:

- Classes responsáveis pela lógica do jogo naquele estado.
- Também são responsáveis por encaminhar o jogo para outros estados.

```
22 usages  14 inheritors  sudo-john-blossom
public abstract class Element implements IMazeElement{

    protected Level level;

    6 usages  sudo-john-blossom
    protected Element(Level level) {
        this.level = level;
    } // o facto de ter aqui o construtor eu depois n tenho que repetir nas classes derivadas

    //abstract public void evolve();

    1 override  sudo-john-blossom
    public boolean move(){return false;}
    no usages  1 override  sudo-john-blossom
    public boolean eat(){return false;}
}
```

Element:

- Classe abstrata que funciona como classe base para todos os elementos do maze.
- Contém implementações por omissão das funções que os elementos vão implementar.

Mobs:

- Classes que definem os mobs e o símbolo que os representa.
- Contém as funções de movimento, etc.

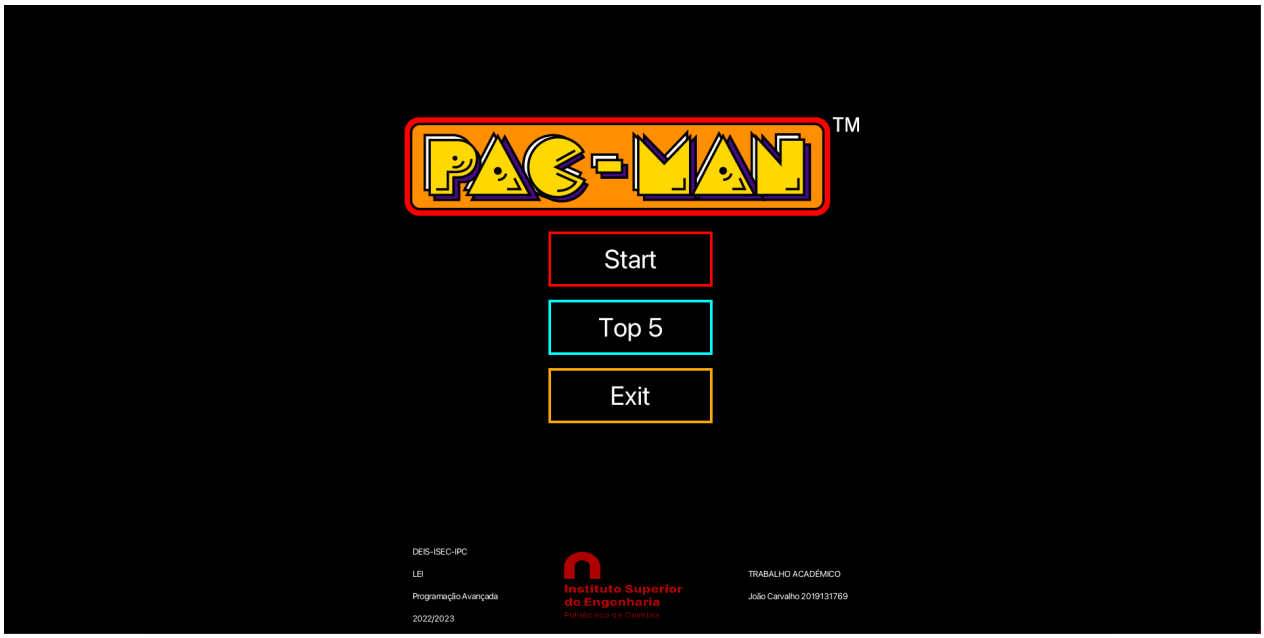
```
public class TinyPac extends Element {
    public static final char SYMBOL = 'M';

    2 usages  sudo-john-blossom
    public TinyPac(Level level) { super(level); }

    sudo-john-blossom
    @Override
    public char getSymbol() { return SYMBOL; }

    sudo-john-blossom
    @Override
    public boolean move(/*KeyEvent e*/){
        Level.Position myPos = level.getPositionOf( element: this);
        level.addElement(new EmptyCell(level), myPos.y(), myPos.x());
        level.addElement(new TinyPac(level), myPos.y(),  x: myPos.x() + 1);
```

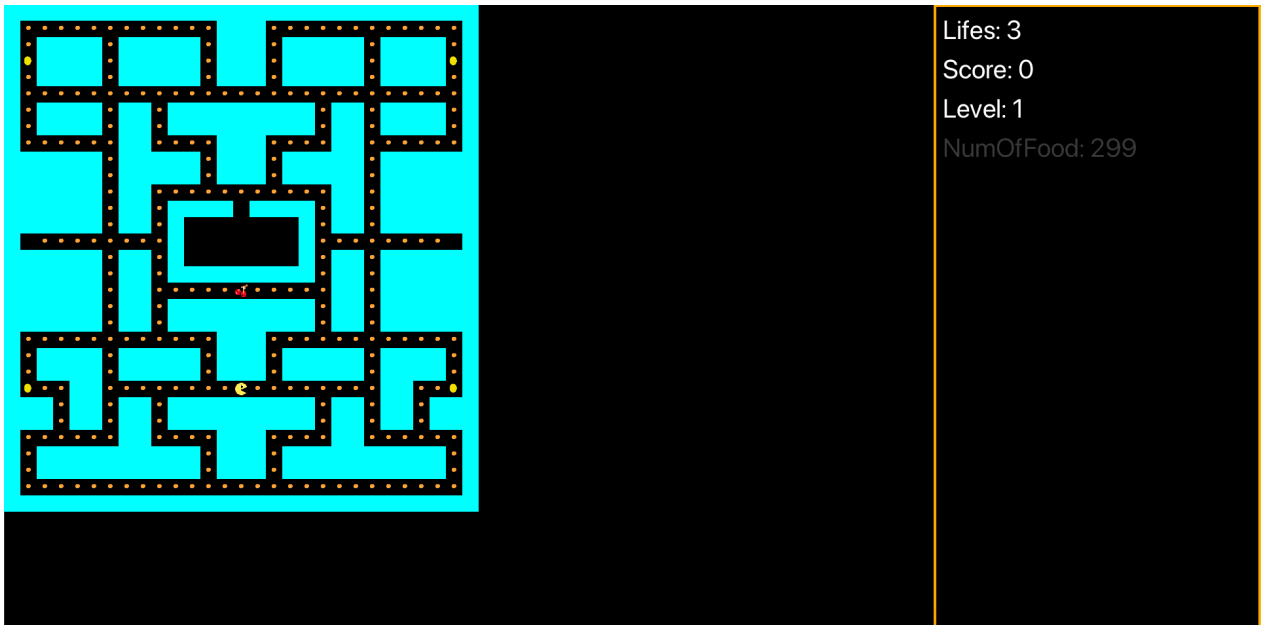
```
public class Cell extends Element {

    1 usage
    private static final char SYMBOL = 'x';

    8 usages  sudo-john-blossom
    public Cell(Level level) { super(level); // depois na herança cada celula vai ter um type diferente }

    8 overrides  sudo-john-blossom
    @Override
    public char getSymbol() { return SYMBOL; }

    /*@Override
    public void evolve(){
    }*/
}
```

Cells:

- Classes que seguem exatamente o mesmo princípio das classes dos Mobs

JavaFX


PAC-MAN
TM
Start
Top 5
Exit
DEIS-ISEC-IPC
LEI
Programação Avançada
2022/2023
Instituto Superior de Engenharia
Politécnico de Coimbra
TRABALHO ACADÉMICO
João Carvalho 2019131769


Lifes: 3
Score: 0
Level: 1
NumOfFood: 299