Relatório Meta 1

Programação Avançada

Instituto Superior de Engenharia Informática

Turma Prática 3

João Carvalho

2019131769

1 de maio de 2023

1. Descrição

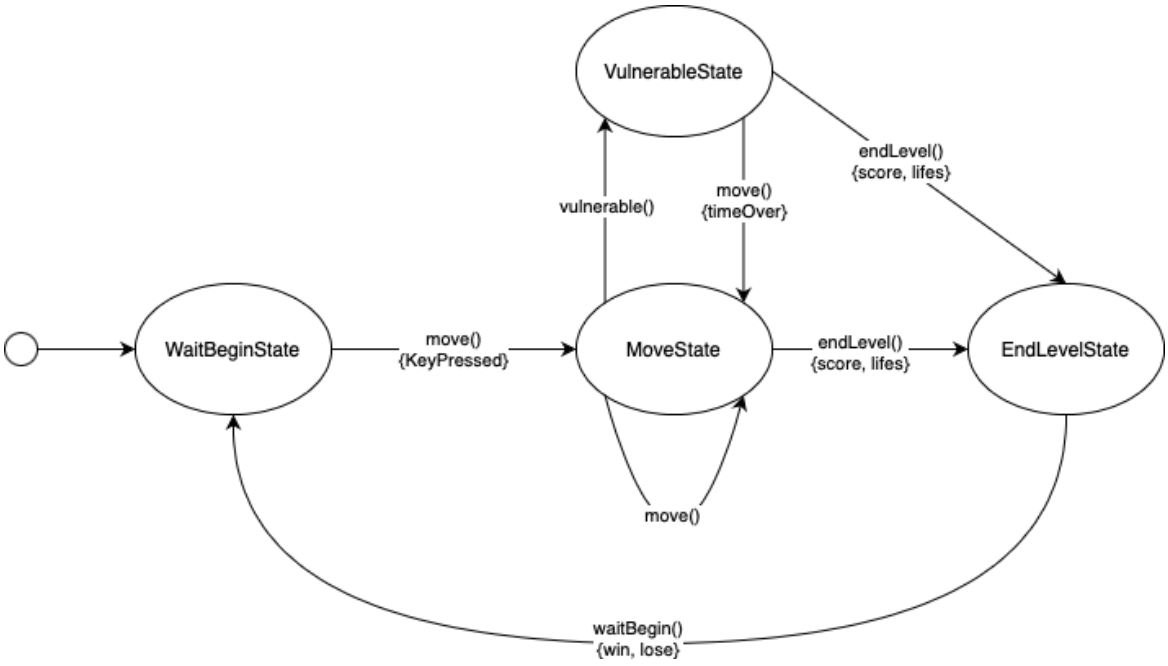
O objetivo deste trabalho foi a implementação de um jogo tipo PacMan.

Apesar de não ter sido implementado na sua totalidade neste meta, 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.

O jogo também é composto por níveis que são geridos por um LevelManager.

Nesta meta foi utilizado a interface da biblioteca Lanterna para imprimir o mapa do nível atual e também foi implementada uma interface de texto onde é possível verificar as transições dos estados do jogo.

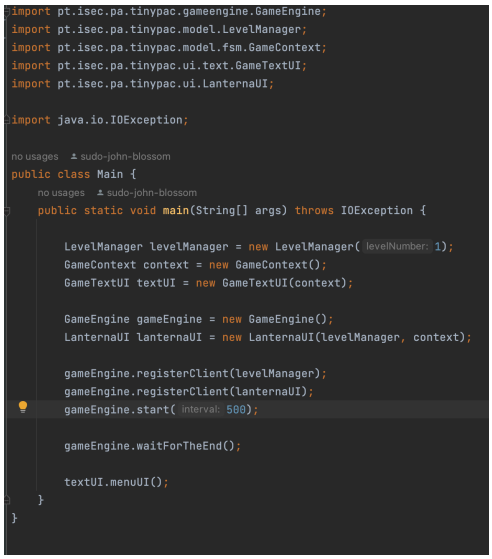
2. Máquina de Estados



Esta máquina de estados é constituída por 4 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.
- EndLevelState: estado do jogo que é iniciado após o jogador atingir o score necessário para passar de nível ou após o jogador ter perdido todas as vidas. Consoante isto o jogo volta para o WaitBeginState com a informação de que nível deve executar, se o mesmo ou o seguinte.

3. Classes



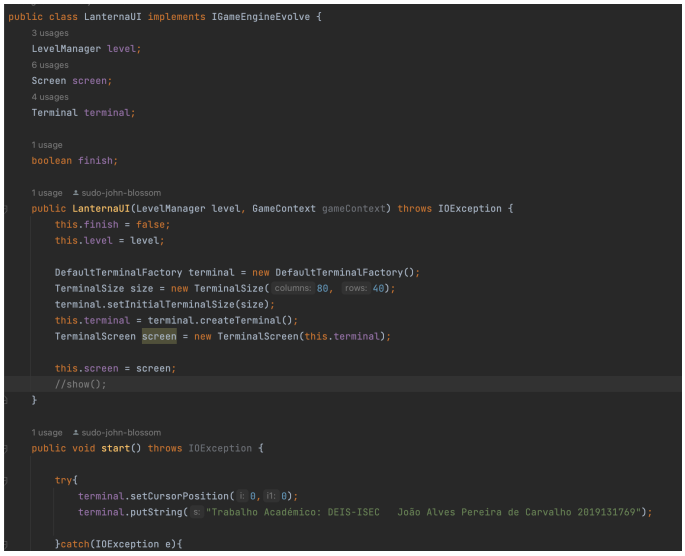
```java
import pt.isec.pa.tinypac.gameengine.GameEngine;
import pt.isec.pa.tinypac.model.LevelManager;
import pt.isec.pa.tinypac.model.fsm.GameContext;
import pt.isec.pa.tinypac.ui.text.GameTextUI;
import pt.isec.pa.tinypac.ui.LanternaUI;

import java.io.IOException;

no usages  sudo-john-blossom
public class Main {
    no usages  sudo-john-blossom
    public static void main(String[] args) throws IOException {

        LevelManager levelManager = new LevelManager( levelNumber: 1);
        GameContext context = new GameContext();
        GameTextUI textUI = new GameTextUI(context);

        GameEngine gameEngine = new GameEngine();
        LanternaUI lanternaUI = new LanternaUI(levelManager, context);

        gameEngine.registerClient(levelManager);
        gameEngine.registerClient(lanternaUI);
        gameEngine.start( interval: 500);

        gameEngine.waitForTheEnd();

        textUI.menuUI();
    }
}
```

Main:

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

LanternaUI:

- Classe onde a interface Lanterna está implementada.
- Responsável por imprimir o mapa do nível atual.



```java
public class LanternaUI implements IGameEngineEvolve {
    3 usages
    LevelManager level;
    6 usages
    Screen screen;
    4 usages
    Terminal terminal;

    1 usage
    boolean finish;

    1 usage  sudo-john-blossom
    public LanternaUI(LevelManager level, GameContext gameContext) throws IOException {
        this.finish = false;
        this.level = level;

        DefaultTerminalFactory terminal = new DefaultTerminalFactory();
        TerminalSize size = new TerminalSize( columns: 80,  rows: 40);
        terminal.setInitialTerminalSize(size);
        this.terminal = terminal.createTerminal();
        TerminalScreen screen = new TerminalScreen(this.terminal);

        this.screen = screen;
        //show();
    }

    1 usage  sudo-john-blossom
    public void start() throws IOException {

        try{
            terminal.setCursorPosition( i: 0, i1: 0);
            terminal.putString( s: "Trabalho Académico: DEIS-ISEC   João Alves Pereira de Carvalho 2019131769");

        }catch(IOException e){
```

```java
public class GameTextUI {

    9 usages
    GameContext gameContextFsm;

    5 usages
    boolean finish;

    1 usage  sudo-john-blossom
    public GameTextUI(GameContext gameContext){
        this.gameContextFsm = gameContext;
        this.finish         = false;
    }

    1 usage  sudo-john-blossom
    public void start(){
        while(!finish){
            switch(gameContextFsm.getState()){
                case WAIT_BEGIN -> waitBeginUI();
                case MOVE -> moveUI();
                case VULNERABLE -> vulnerableUI();
                case END_LEVEL -> endLevelUI();
                //case NEXT_LEVEL -> nextLevelUI();
            }
        }
    }

    2 usages  sudo-john-blossom
    public void menuUI(){
```

GameTextUI:

- Classe responsável pela criação da interface de texto.

```java
public class LevelManager implements IGameEngineEvolve {

    1 usage
    private int levelNumber;
    5 usages
    Level level;

    2 usages   sudo-john-blossom
    public LevelManager(int levelNumber){
        this.levelNumber = levelNumber;
        String filePath = "files/Level10" + levelNumber + ".txt";
        this.level = setMap(filePath);
    }

    1 usage   sudo-john-blossom
    public Level setMap(String filePath) {

        try{
            char[][] auxMap;

            File file = new File(filePath);
            if(!file.exists())
                return null;
            Scanner sc = new Scanner(file);

            String firstLine = sc.nextLine();
            int height = 1;
            int width = firstLine.length();

            while(sc.hasNextLine()){
                height++;
                sc.nextLine();
            }
```

LevelManager:

- Classe responsável por gerir o nível em que o jogo se encontra.
- É também responsável por ler o maze do ficheiro de texto e atribui-lo a um nível.

MobsStateAdapter:

- Classe abstrata responsável por mudar o estado em que o jogo se encontra e implementar funções por omissão dos métodos da interface IMobsState.

```java
public abstract class MobsStateAdapter implements IMobsState {

    2 usages
    protected Game game;

    3 usages
    protected GameContext context;

    4 usages   sudo-john-blossom
    protected MobsStateAdapter(GameContext context, Game game){
        this.context    = context;
        this.game       = game;
    }

    7 usages   sudo-john-blossom
    protected void changeState(EMobsState newState){context.changeState(newState.createState(context, game));}

    3 overrides   sudo-john-blossom
    @Override
    public boolean move(){return false;}

    1 usage   1 override   sudo-john-blossom
    @Override
    public boolean vulnerable(){return false;}

    1 usage   3 overrides   sudo-john-blossom
    @Override
    public boolean endLevel(){return false;}
}
```

```java
public interface IMobsState {

    4 implementations   sudo-john-blossom
    boolean move();
    1 usage   2 implementations   sudo-john-blossom
    boolean vulnerable();
    1 usage   4 implementations   sudo-john-blossom
    boolean endLevel();

    1 usage   4 implementations   sudo-john-blossom
    EMobsState getState();
}
```

IMobsState:

-Interface que contém as funções que são implementadas nos estados.

```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.

EMobsState:

- Enumeração que contem os estados do jogo e uma fábrica de objetos que é responsável por criar os estados.

```java
public enum EMobsState {

    //MENU, WAIT_BEGIN, PLAYING_LEVEL, PAUSED, GAMEOVER_MENU, WIN_MENU/*, NEXT_LEVEL*/;

    //FRABRICA DE OBJETOS
    /*IMobsState createState(GameContext context, Game game){
        return switch(this){
            case MENU -> new MenuState(context, game);
            case WAIT_BEGIN -> new WaitBeginState(context, game); //neste estado ele espera que o utilizador pressione uma tec
            case PLAYING_LEVEL -> new PlayingLevelState(context, game); //neste estado o jogo esta a ser jogado
            case PAUSED ->  new PausedState(context, game); //neste estado o jogo encontra-se em pausa
            case WIN_MENU -> new WinMenuState(context, game); //neste estado esta a ser exibido o menu de vitoria
            case GAMEOVER_MENU -> new GameOverMenuState(context, game); //neste estado esta a ser exibido o menu de derrota
            //case NEXT_LEVEL -> new NextLevelState(context, game); //neste estado muda o nivel atual
        };

    }*/

    4 usages
    WAIT_BEGIN, MOVE, VULNERABLE, END_LEVEL;

    1 usage  sudo-john-blossom
    IMobsState createState(GameContext context, Game game){
        return switch(this){
            case WAIT_BEGIN -> new WaitBeginState(context, game);
            case MOVE -> new MoveState(context, game);
            case VULNERABLE -> new VulnerableState(context, game);
            case END_LEVEL -> new EndLevelState(context, game);
        };
    }
}
```

```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(){
    }*/
}
```

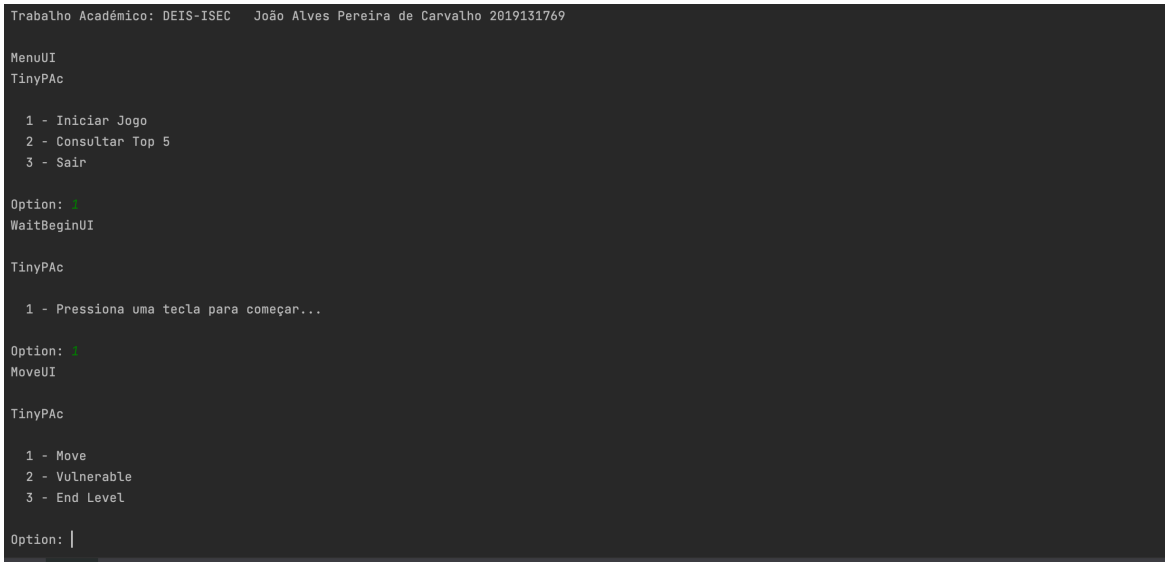
Cells:

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

LanternaUI

Interface utilizada para representar a maze.

TextUI



Interface utilizada para verificar a mudança de estados da máquina de estados.