

# PROGRAMAÇÃO AVANÇADA

ANNA BEATRIZ YABE

2021132515

**META 2**

---

# SUMÁRIO

<b>INTRODUÇÃO .....</b>	<b>3</b>
<b>DECISSÕES NA IMPLEMENTAÇÃO.....</b>	<b>4</b>
<b>DIAGRAMA DA MÁQUINA DE ESTADOS</b>	<b>5</b>
<b>CLASSES .....</b>	<b>7</b>
<b>RELACIONAMENTOS ENTRE CLASSES</b>	<b>13</b>
<b>FUNCIONALIDADES .....</b>	<b>14</b>
<b>TESTES UNITÁRIOS .....</b>	<b>16</b>

---

# INTRODUÇÃO

O presente trabalho tem como objetivo o desenvolvimento do jogo Pac-Man (versão Tiny-Pac) para a disciplina de Programação Avançada no ano letivo 2022-2023, na licenciatura de Engenharia Informática do Instituto Politécnico de Coimbra.

Para a primeira meta do trabalho foram implementados componentes que garantem o breve funcionamento e teste dos estados da Final State Machine (fsm).

Já a segunda meta englobou mais funcionalidades para permitir o funcionamento do jogo. Entre elas o desenvolvimento de uma interface gráfica recorrendo ao uso do JavaFX.



# DECISSÕES NA IMPLEMENTAÇÃO

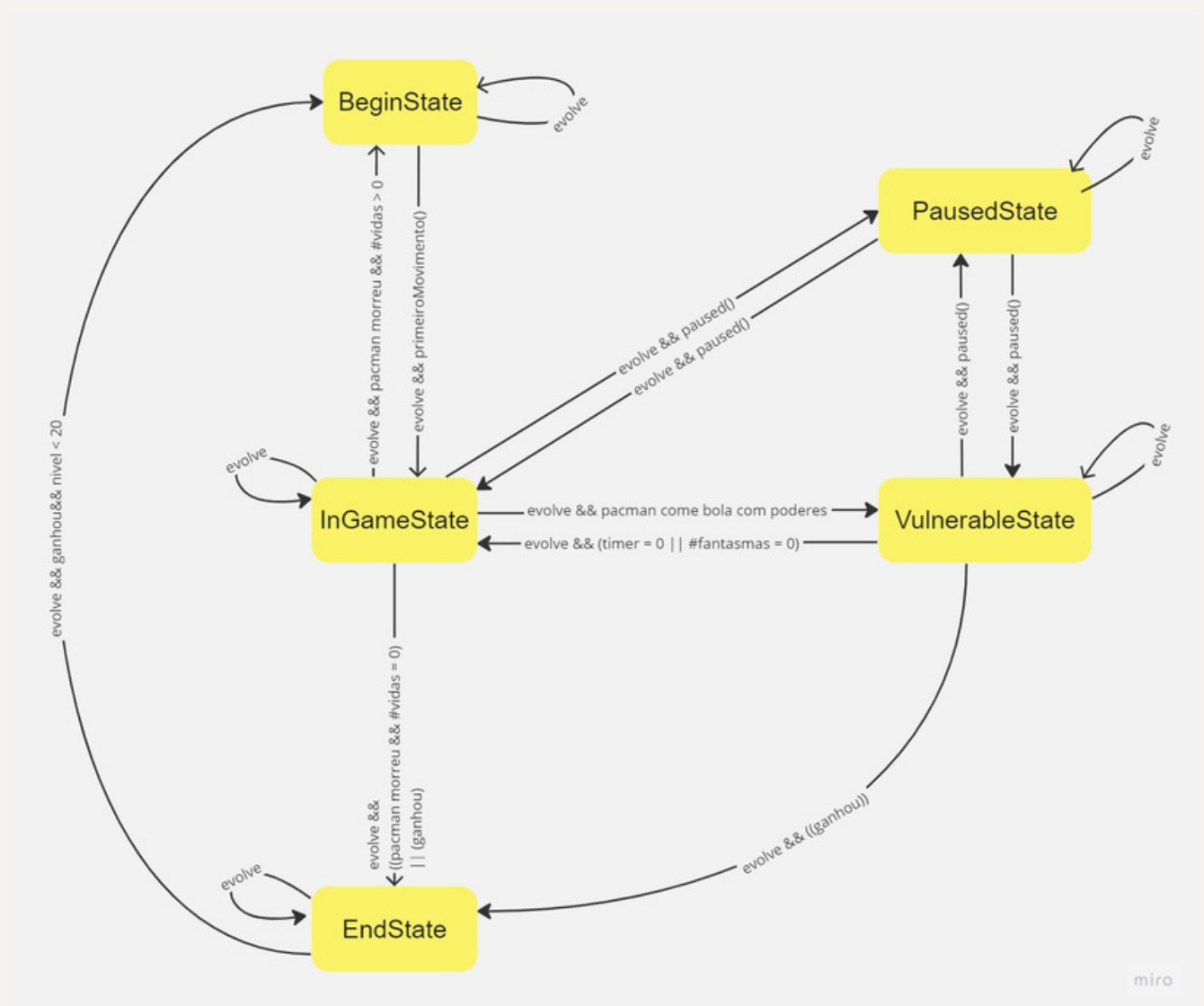
Ao longo da realização desse trabalho, foram realizadas escolhas que estarão brevemente descritas nessa parte do relatório.

No quesito de classes, é importante salientar a classe `Game`. Esta consiste em uma classe voltada para gerenciar o jogo, o que significa que ela armazena as variáveis que guardam informações sobre o jogo. Variáveis que indicam por exemplo, se o jogo esta no modo vulnerável, se o jogo foi ganho ou perdido, se o jogo está em pausa. Existem também variáveis essenciais para guardar o número de vidas do Pacman (3), quantidade de tempo passado desde o início do jogo e pontos acumulados.

Ainda sobre o modelo de dados, pode se dizer que três classes são de extrema importância o `Game`, `Context` e `GameManager`. O `Context` é a classe destinada a permitir que as modificações no modelo de dados ocorram apenas no seu respectivo contexto. Enquanto a classe `GameManager` é a classe que gerencia o modelo de dados e simultaneamente atualiza as vistas. Ela utiliza do padrão observable para que as mudanças das vistas sejam de maneira assíncrona.

O jogo permite entrar em um estado vulnerável, onde foi determinado que a sua duração é de 40 segundos, podendo esse valor ser alterado no código. E neste estado o PacMan fica impedido de comer uma nova bola com poderes. Ainda quanto aos elementos do tabuleiro, quando o PacMan passa sobre a `ZonaWrap` e é transportado ao outro lado do tabuleiro, ele fica impedido de passar pela `ZonaWrap` por alguns instantes. De modo a impedir que ele fique em um ciclo infinito indo e voltando.

# DIAGRAMA DA MÁQUINA DE ESTADOS



## BREVE DESCRIÇÃO DOS ESTADOS

**BeginState:** estado no qual o utilizador selecionou para iniciar o jogo, mas ainda não realizou nenhum movimento do PacMan. Quando o utilizador pressiona alguma tecla de movimento, segue para o próximo estado (`primeiro_movimento()` -> `InGameState`).

**InGameState:** estado de funcionamento normal do jogo, em outras palavras, quando não está em pausa, nem vulnerável e o jogo não foi encerrado. É neste estado que é chamado o método `evolve()`, responsável pelo movimentos dos **Agents**. Os estado seguintes podem ser: `PausedState`, `VulnerableState` ou `EndState`.

**PausedState:** estado no qual o jogo se encontra parado, ação ativada ao ser pressionado o botão. É verificado se o jogo esta no modo vulnerável, caso não esteja o seu estado seguinte é o `InGameState`, nas caso esteja no modo vulnerável o próximo estado é `VulnerableState`.

**VulnerableState:** estado no qual o jogo tem comportamento diferenciado, do tipo vulnerável. Seus estados seguintes podem ser: `InGameState`, `PausedState` ou ainda `EndState`.

**EndState:** estado no qual o nível terminou seja por vitória ou derrota do jogador.

# CLASSES

## ► PT.ISEC.PA.TINY PAC.MODEL.DATA

### Agents

- classe base para os agentes e implementa a interface IMazeElement;
- dedicada a guardar informações em comum de todos os agentes;

### Blinky

- classe derivada de Agents;
- dedicada a guardar informações e ações que serão realizadas pelo fantasma Blinky;

### Clyde

- classe derivada de Agents;
- dedicada a guardar informações e ações que serão realizadas pelo fantasma Clyde;

### Direção

- enumeração com as possíveis direções que o PacMan pode seguir;

### Game

- dedicada a guardar informações referente ao jogo, como o estado atual no qual ele se encontra, se o jogo se encontra no modo vulnerável, se o jogo terminou e se o jogo esta em pausa;

### GameManager

- classe destinada a gerenciar o modelo de dados, permite que sejam realizadas mudanças nas vistas de maneira assíncrona através do padrão observável;

### IMazeElement

- classe fornecida;

### **Inky**

- classe derivada de Agents;
- dedicada a guardar informações e ações que serão realizadas pelo fantasma Inky;

### **Maze**

- classe fornecida;

### **ModelLog**

- classe que utiliza do padrão singleton para facilitar manter o progresso, permitindo escrever mensagens de controle;

### **PacMan**

- classe derivada de Agents;
- dedicada a guardar informações e ações que serão realizadas pelo PacMan;

### **Pinky**

- classe derivada de Agents;
- dedicada a guardar informações e ações que serão realizadas pelo fantasma Pinky;

### **Players**

- classe para manter informações dos jogadores;

### **ReadMaze**

- responsável pela leitura dos ficheiros de texto (níveis do jogo) e preencher de maneira inicial o Maze;

### **Top5Players**

- destinada a manter os 5 melhores jogadores;

► **PT.ISEC.PA.TINYPAC.MODEL.DATA.MAZELEMENTS**



### **MazeElementBola**

- implementa a interface IMazeElement;
- representa um dos objetos que podem ser encontrados no jogo;

### **MazeElementBolaComPoderes**

- implementa a interface IMazeElement;
- representa um dos objetos que podem ser encontrados no jogo;

### **MazeElementCaverna**

- implementa a interface IMazeElement;
- representa um dos objetos que podem ser encontrados no jogo;

### **MazeElementFruta**

- implementa a interface IMazeElement;
- representa um dos objetos que podem ser encontrados no jogo;

### **MazeElementParede**

- implementa a interface IMazeElement;
- representa um dos objetos que podem ser encontrados no jogo;

### **MazeElementPosicaoInicialFantasmas**

- implementa a interface IMazeElement;
- representa um dos objetos que podem ser encontrados no jogo;

### **MazeElementPosicaoInicialPacMan**

- implementa a interface IMazeElement;
- representa um dos objetos que podem ser encontrados no jogo;

### **MazeElementZonaWrap**

- implementa a interface IMazeElement;
- representa um dos objetos que podem ser encontrados no jogo;

### **BeginState**

- classe derivada de StateAdapter;
- representa o estados da FSM quando o jogo ainda não começou, mas já mostra o tabuleiro;

### **Context**

- classe que garante que as mudanças só podem ocorrer no contexto de métodos relacionados com mudanças de estados

### **EndState**

- classe derivada de StateAdapter;
- representa o estados da FSM quando o jogo termina;

### **InGameState**

- classe derivada de StateAdapter;
- representa o estados da FSM quando o jogo esta em andamento;

### **IState**

- interface que apresenta os métodos que representam todas as transições;

### **PausedState**

- classe derivada de StateAdapter;
- representa o estados da FSM quando o jogo esta em pausa;

### **StateAdapter**

- implementa a interface IState;
- classe que apresenta implementações por defeito para todas as transições e um método que permite mudar de estado

### **States**

- enumeração que permite a identificação de todos os estados;

## VulnerableState

- classe derivada de StateAdapter;
- representa o estados da FSM quando o jogo esta em modo vulnerável;

## PT.ISEC.PA.TINYPAC.GAMEENGINE

# Game Engine

- classe fornecida;

## IGameEngineState

- classe fornecida;

# I Game Engine

- classe fornecida;

# IGameEngineEvolve

- classe fornecida;

## PT.ISEC.PA.TINYPAC.UI.TEXT

## MenuText

- classe destinada aos `System.out.print` que representam o menu principal do jogo;

## PAInput

- classe fornecida nas aulas práticas para definir menu de texto e obter input do utilizador;

## TextUI

- classe para mudar com menu de texto os estados (FSM);

## PT.ISEC.PA.TINYPAC.UI.GUI

## BeginStateUI

- interface para o estado BeginState;

## EndStateUI

- interface para o estado EndState;

### **ExitUI**

- interface para o quando o jogo termina;

### **InfoGame**

- painel de informações sobre o jogo corrente;

### **InGameUI**

- interface para o estado InGameState;

### **MainMenuUI**

- interface inicial do menu;

### **MazeUI**

- apresenta um gridPane com o tabuleiro do jogo;

### **PausedStateUI**

- interface para o quando o jogo está em pausa;

### **RegisterPlayerUI**

- interface para registar um novo jogador;

### **ResumeUI**

- interface para quando é feito um load de um jogo salvo;

### **RootPane**

- representa o root pane;

### **Top5UI**

- interface para mostrar o TOP 5;

### **VulnerableStateUI**

- interface para o jogo no estado vulnerável;

## **PT.ISEC.PA.TINYPAC.UI.GUI.RESOURCE**

### **ImageLoader**

- classe fornecida nas aulas teóricas e práticas para colocar imagens nas vistas;



# FUNCIONALIDADES

Funcionalidades	Situação Atual	Observações
Implementação do jogo com uma interface com o utilizador (IU) em modo gráfico (JavaFX) A atualização desta interface, bem como outras atualizações de informação, deverá estar de acordo com o padrão de notificações assíncronas estudado nas aulas	Parcialmente implementado	O jogo não possui todas as funcionalidades descritas no enunciado. Mas apresenta uma interface gráfica que utiliza do JavaFX e do padrão de notificações assíncronas estudado.
Implementação do Top 5, incluindo a criação, registo de novo resultado, persistência (serialização) e apresentação através de interface gráfica	Totalmente implementado	O TOP 5 é apresentado numa vista e é atualizado somente quando o jogador pontua um valor maior que outros. Entretanto, ao final de todos os jogos é requerido o nome do jogador (mesmo que ele não vá para o top5), sendo a atualização deste somente com a necessidade.
As interfaces com o utilizador básicas que não sejam significativamente distintas de uma versão em modo texto serão bastante penalizadas. Espera-se que as interfaces com o utilizador sejam apelativas, intuitivas e funcionais	Totalmente implementado	As interfaces diferem-se da parte de lógica do jogo. Sendo a user interface amigável e de fácil compreensão.

Funcionalidades	Situação Atual	Observações
<b>Gravação e restauro do jogo em ficheiro binário através do processo de serialização</b>	Parcialmente implementado	Algumas vezes, acontece um erro ao salvar o ficheiro binário. A princípio, com a mudança do nome do ficheiro já permite que não exista esse problema mais.
<b>Código devidamente comentado usando JavaDoc</b>	Totalmente implementado	Foi utilizado de comentários ao longo principalmente do modelo de dados para posteriormente gerar um JavaDoc.
<b>Incorporação de testes unitários nas classes relativas à máquina de estados</b>	Totalmente implementado	Foram realizados testes unitários na máquina de estados testando as suas classes (tópico a seguir).


# TESTES UNITÁRIOS

Foram utilizados testes unitários para garantir o comportamento correto da máquina de estados. O principal objetivo era verificar se a máquina de estados funciona da maneira esperada. Dessa forma, é possível evitar comportamentos inesperados.


Para a realização destes testes foi criada um novo módulo chamado "Testes". A fim de evitar erros de não encontrar os ficheiros de texto, foram adicionados a esse módulo os ficheiros básicos para o funcionamento do jogo.

Abaixo estão listados todos os testes feitos bem como os seus respectivos resultados e objetivos do teste. Todos estes testes foram efetuados na classe "Tests".

## BeginState\_teste\_primeiroMovimentoUp()

- verifica se no estado BEGINSTATE é pressionado uma tecla de movimento resulta em mudança de estado para INGAMESTATE
- `assertEquals(EM_JOGO, estado);` 

## BeginState\_teste\_pausa()

- verifica se no estado BEGINSTATE é possível colocar o jogo no estado PAUSESTATE (não deve ser permitido, por isso mantem-se no mesmo estado)
- `assertEquals(INICIO, estado);` 

## BeginState\_teste\_vulneravel()

- verifica se no estado BEGINSTATE é possível colocar o jogo no estado VULNERABLESTATE (não deve ser permitido, por isso mantém-se no mesmo estado)
- assertEquals(INICIO, estado);

## InGameState\_teste\_pausa()

- verifica se no estado `INGAMESTATE` é possível colocar o jogo no estado `PAUSESTATE`
- ```
assertEquals(EM_PAUSA, estado);
```
- 