# Sokoban Game – Relatório

## Sokoban Game

No âmbito da unidade curricular de Programação Orientada para Objetos, fomos desafiados a desenvolver um jogo de Sokoban, que é um jogo de estratégia que consiste, resumidamente, na existência de um *player* que deve empurrar determinados objetos específicos para cima de um alvo.

### Manual de Utilização

Quando se inicia o jogo, aparece uma janela com as instruções básicas de jogo (por exemplo, que devem ser utilizadas as setas do teclado para mover o *player*) e algumas opções que poderão ser selecionadas no decorrer do jogo, nomeadamente:

- H – Para abrir uma janela de ajuda (com mais indicações e detalhes sobre os objetos do jogo);
- R – Para recomeçar o jogo (no nível 1) e salvar a pontuação atual[1];
- Q – Para fechar o jogo e salvar a pontuação atual[1].

Ainda nesta janela, o *player* pode selecionar 'OK' para prosseguir ou 'Cancel' para cancelar e não abrir o jogo.

Caso prossiga e não coloque um valor no nome do *player*, o jogo apontará *NullPointerException()*, sendo obrigatória a colocação de um nome. Uma vez isto resolvido, abre a janela do nível 0 – tutorial.

Ao longo de todo o jogo, poderá encontrar-se vários desafios e vários objetos. Podendo estes ter influência na score e/ou energia, de acordo com a tabela seguinte:

*Tabela 1 – Influência nas scores e energia*

<table>
<tr><th>Objeto</th><th>Interage com…</th><th>Influência na score</th><th>Influência na energia</th></tr>
<tr><td>Alvo</td><td>Caixote</td><td>+50</td><td>S/ influência</td></tr>
<tr><td>Bateria</td><td>Player</td><td>+25</td><td>+20</td></tr>
<tr><td rowspan="3">Buraco</td><td>Player</td><td colspan="2">S/ influência<br>Efeito: Perde o jogo</td></tr>
<tr><td>Pedra Grande (efeito: a pedra fica a bloquear o buraco)</td><td rowspan="2">+15</td><td>-10</td></tr>
<tr><td>Qualquer outro objeto movível à exceção dos dois acima (efeito: o objeto desaparece)</td><td>S/ influência</td></tr>
<tr><td rowspan="2">Gelo</td><td rowspan="2">Objetos movíveis</td><td colspan="2">S/ influência</td></tr>
<tr><td colspan="2">Efeito: Apenas faz com que o objeto deslize por cima dele</td></tr>
<tr><td>Martelo</td><td>Player</td><td>+5</td><td>S/ influência</td></tr>
<tr><td rowspan="2">Parede Partida</td><td rowspan="2">Martelo</td><td>+20</td><td>-5</td></tr>
<tr><td colspan="2">Efeito: Parede partida desaparece</td></tr>
<tr><td>Teleporte</td><td>Objetos movíveis</td><td>S/ influência</td><td>-2</td></tr>
</table>

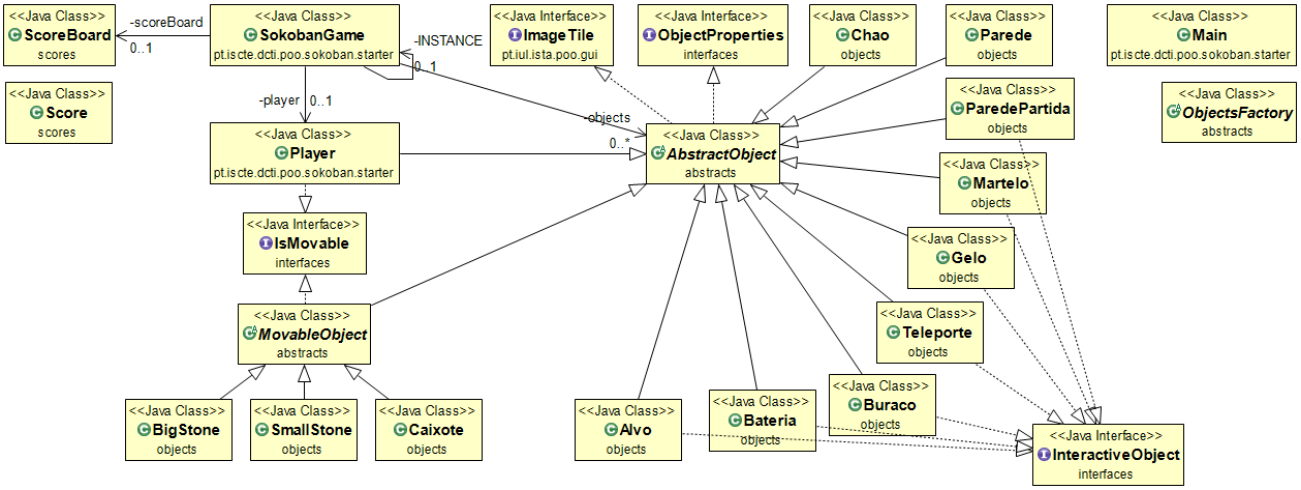
[1] Nestes casos a pontuação do nível é sempre salva no ficheiro de pontuações do nível, a *score* (total) no ficheiro de *HighScores* e, se se encontrar entre os 5 melhores, no ficheiro de Top5, que corresponde às 5 melhores pontuações obtidas no jogo.

Além disso, ao passar de nível, acresce um bónus de 100 pontos e energia restante.

# Desenho de classes

No desenvolvimento deste jogo, procurou-se simplificar ao máximo o código e torná-lo, simultaneamente, escalável. Para isso, a estrutura de classes é a que se pode verificar na figura 1.

*Figura 1 – Desenho UML das classes*




## Objetos de Jogo

### A classe *AbstractObject*

Todos os objetos de jogo têm como base a classe *AbstractObject*, em que os mesmos são criados e ganham atributos como o **nome** (que servirá de indicador para a imagem a utilizar), a **posição** (Point2D) e a ***layer*** ("camada" do jogo). Esta classe é composta por três construtores que serão herdados pelos objetos:

1. Aceita todos os atributos no método construtor (nome, posição, *layer*);
2. Aceita dois atributos no método construtor (nome e posição) e assume o outro como um *default* (layer igual a 1);
3. Aceita um atributo (posição), que é utilizado, sobretudo, na fábrica de objetos para criação de níveis.

Esta classe implementa, ainda, a interface *ObjectProperties*, pré-definindo certas características:

- *boolean transposableObject(IsMovable obj) = true*
- *boolean movableObject() = false*
- *boolean interactiveObject() = false*

Os objetos de jogo podem, assim, dividir-se em três categorias:

- **Estáticos** como o *Chão* e a *Parede*;
- **Movíveis** que implementam a interface *IsMovable*;
- **Interativos** que implementam a interface *InteractiveObject*.

### Objetos Estáticos

Existem apenas dois objetos estáticos e que não são interativos: o *Chão* e a *Parede*. A única diferença entre estes dois objetos é que o objeto *Chão* é *transposableObject(IsMovable obj) = true* e o objeto *Parede* é *= false*.

### A interface *IsMovable*

Os objetos **movíveis**, como é o caso do *Player*[2] e dos objetos de classes derivadas de *MovableObject*, implementam esta interface, em que são definidos cinco métodos:

- *boolean isMovable(Direction direction)* – verifica se o objeto se pode mover;
- *void move(Direction d)* – efetua o movimento do objeto;
- *void fall()* – define como é que o objeto cai (p.e. no *Buraco*);
- *boolean getFall()* – verifica se o objeto cai ou não (p.e. no caso da *PedraGrande*, é dado *@Override* ao método definido em *MovableObject*, porque é um caso específico);
- *Direction direction()* – devolve a direção do movimento do objeto.

É importante referir que é no *move(Direction d)* que é efetuada a função de interação com os objetos interativos e que existem *ObjectProperties* em que é dado *@Override* como o *transposableObject(IsMovable obj) = false* e o *movableObject() = true*.

Optou-se por criar uma classe abstrata *MovableObject*, porque a maioria dos métodos definidos por esta interface são comuns a todos os objetos movíveis, à exceção do *Player*, que é um caso especial. Isto evitou muita repetição de código.

### A interface *InteractiveObject*

Os objetos **interativos** (como os indicados na figura 1 que implementam a interface *InteractiveObject*) devem definir em cada um deles cinco métodos:

- *boolean isInteracting(IsMovable obj)* – verifica se está a existir interação com o objeto movível que cai em cima do próprio objeto interativo e invoca a interação;
- *void interaction(IsMovable obj)* – define a interação com o respetivo objeto movível;
- *void dialogue()* – envia um *System.out.println()* para a consola;
- *int scoreInfluence()* – define a influência na score do jogo;
- *int energyInfluence()* – define a influência na energia do nível.

Em todos os objetos que implementam esta interface foi dado *@Override* sobre o método *interactiveObject() = true* definido em *ObjectProperties*.

Ao contrário daquilo que foi feito com os objetos movíveis, no caso dos objetos interativos, não se justificou a criação de uma classe abstrata para os mesmos, uma vez que acabava por não existir nenhum método que fosse inteiramente comum a todos e existia apenas a necessidade de estes métodos estarem definidos em cada um destes objetos.

### A classe ScoreBoard

Esta classe tem apenas a função de guardar as *scores* por nível no respetivo ficheiro e, adicionalmente, de armazenar as cinco melhores scores do jogo num Top5. Esta classe permite também a criação de um Top3 que é apresentado no fim de cada nível (com as *scores* do mesmo à exceção da do jogo a decorrer) e de um Top3 das *HighScores* no último nível, quando se vence o jogo (em que a score atual já é incluída).

[2] O *Player* é um caso especial de objeto, pois representa o elemento controlado pelo jogador.

As *scores* são ordenadas por ordem decrescente (do melhor para o pior) de acordo com um *compareTo(Score scoreData)* definido na classe *Score*.

## Declaração de Honra

Eu, Mara Andreia Pimpão Alves, nº de aluna 94013, da turma 1 do primeiro ano de licenciatura em Engenharia de Telecomunicações e Informática, atesto que o código entregue neste trabalho é integralmente de minha autoria.