







Programação Orientada por Objetos

Primeiro trabalho prático
Semestre de Verão 2018/2019

Objetivos: Neste trabalho pretende-se que os alunos adquiram prática de utilização de herança e polimorfismo usando a abordagem MVC na implementação de um jogo em consola de texto.

Entrega: Cada grupo entregará no *moodle* da sua turma, até dia 8 de abril, os ficheiros fonte (java) do projeto realizado e o documento que descreve a implementação do trabalho contendo o diagrama de classes.

Enunciado: Implemente o jogo designado por **Sokoban** cujo objetivo é deslocar a pessoa (representado por ) dentro de um armazém, para empurrar todos os caixotes (representados por ) para os seus destinos , passando a representação do caixote a . As paredes do armazém  e os buracos  não se deslocam. O jogo termina se a pessoa cair num buraco, mas os caixotes podem ser empurrados para os buracos, passando essa posição a ser chão normal.

A figura 1 apresenta o aspeto de um estado possível deste jogo.

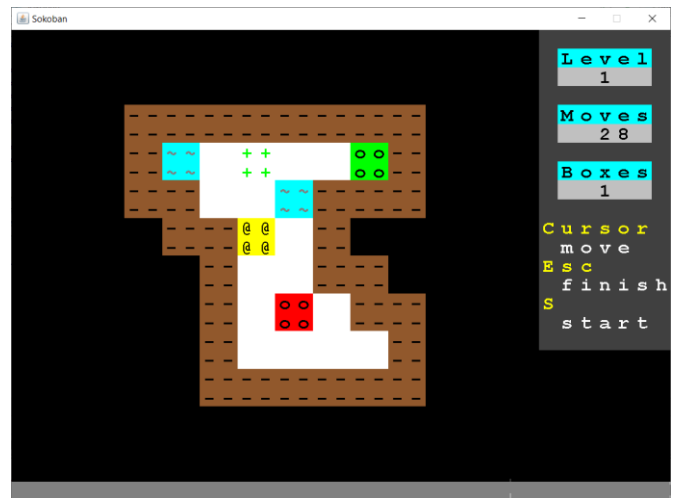


Figura 1: Aspeto do jogo

Os níveis são descritos no ficheiro de texto (levels.txt). A primeira linha de cada nível indica o número e as dimensões do armazém (linhas x colunas) e as restantes linhas descrevem o estado inicial. Cada símbolo tem um significado ('@'-Pessoa; 'B'-Caixote; '.'-Vazio; '-'-Chão; 'X'-Obstáculo; '*'-Objetivo; 'H'-Buraco). Uma letra minúscula (neste exemplo 'a') indica que a posição tem mais do que um símbolo que são descritos no final (neste caso '*' e 'B').

A figura 2 apresenta o troço do ficheiro que descreve o estado inicial do armazém da figura 1.

```
#1 8 x 8
XXXXXXXXX
XH a *X
XX HXXX
.XX X..
..X BXX.
..X XX
..X @X
..XXXXXX
a=*B
```

Figura 2:
Troço de levels.txt

O programa deve construir cada nível a partir da descrição do ficheiro e deslocar a pessoa de acordo com as teclas de direção premidas, empurrando os caixotes, caso seja possível. O jogo passa para o nível seguinte quando todos os caixotes ocuparem os objetivos. O jogo termina com sucesso se forem completados todos os níveis descritos no ficheiro. O jogo termina sem sucesso se a pessoa cair num buraco ou se for premida a tecla 'Esc'. O nível corrente é reiniciado quando premida a tecla 'S'.

Pode ser experimentada uma versão do jogo pretendido em "[Sokoban.zip](#)".

Este *zip* contém um *jar* com o programa de demonstração, o ficheiro de descrição dos níveis, o *jar* da biblioteca ConsolePG e uma pasta "src" com os ficheiros fonte de algumas das classes já implementadas. Destas classes também é possível depreender algumas das classes que são necessárias definir e alguns dos métodos públicos.

A figura 3 mostra as classes principais do programa, apresentando em fundo azul as totalmente implementadas. Em fundo amarelo estão representadas as hierarquias de classes que devem ser definidas para cada tipo de célula, no modelo e na visualização, e para cada tipo de ator (elementos que se deslocam nas células).

Implemente o jogo descrito seguindo a abordagem MVC.

1. No **modelo**, defina as hierarquias de classes que representem os vários tipos de células que descrevem as características do mapa do armazém (classe *Cell* e suas derivadas) e os vários tipos de atores que se deslocam no armazém (classe *Actor* e suas derivadas) e defina a classe *Level* que implementa a lógica do jogo (as regras) em cada nível. O código destas classes será reutilizado na implementação deste jogo em *Android* sem necessidade de alterações.
2. A componente de **controlo** que interage com o utilizador (classe *Sokoban*), processa as teclas. O **controlo** recebe notificações e chama métodos do **modelo** para evoluir no estado do jogo e métodos da componente de **visualização** para redesenhar as células.
3. A **visualização** usa a classe *Console*, da biblioteca ConsolePG incluída no *zip*, e as classes do package *isel.poo.console*, faltando definir as classes *StatusPanel*, *CellTile* e derivadas que implementam a apresentação de cada tipo de célula.

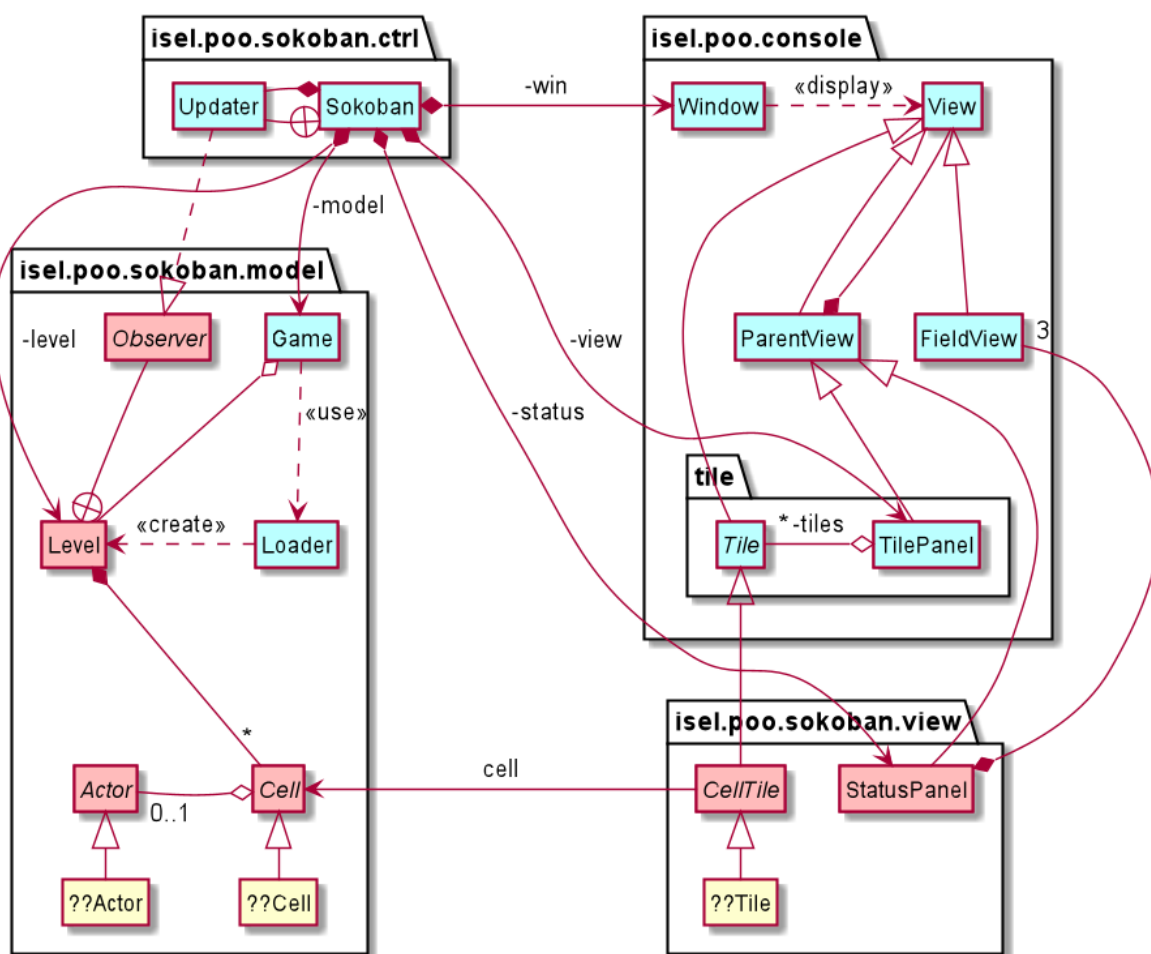


Figura 3: Diagrama de classes

Opcionalmente poderão ser implementados outros tipos de células e atores, por exemplo:

- Caixotes leves que podem ser empurrados em conjunto.
- Zonas do armazém em que os caixotes e a pessoa só podem passar num sentido.
- Portas que apenas se abrem quando se empurra a chave até elas.
- Etc.