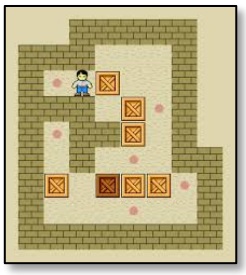
Course 'Imperative Programming' (IPC031)



Assignment 13: Sokoban

1. Background

Sokoban[1] is a sliding puzzle created by Hiroyuki Imabayashi in 1981. In such a puzzle, the world can be represented by a matrix of cells. The world is surrounded by walls, and there can be more walls inside the world as well. There is one person, the *warehouse keeper*, who has the task to move the present boxes to *destination cells*. A destination cell is marked (in the image to the right by the pink spots). The warehouse keeper can move only one cell at a time to the right, left, up, or down. Neither the warehouse keeper nor boxes can be moved into a wall. The warehouse keeper can only push a box, not pull or lift it. The warehouse keeper is not strong enough to push more than one box at the same time. The puzzle is finished if all boxes are on a destination cell.

2. Learning objectives

After doing this assignment you are able to:

- implement breadth-first and depth-first search problems;
- apply search problem solving techniques for new search problems.

3. Assignment

On Brightspace, you find the file *"IPC031_2018_assignment_13_files.zip"*. It contains a number of text files. The number in the file name is the number of steps to solve the challenge. So, *challenge.0* is already a final solution. Note that *challenge.34* (the above puzzle) required *one and a half hour* to compute on a Windows 10 laptop (2.5GHz, 6GB RAM, Intel i5-3210M CPU). In these text files a puzzle starting configuration is described by a matrix of cells that have the following representation:

- a wall cell by `'*'`,
- an empty cell by `' '` (space character),
- an empty destination cell by `'.'`,
- the worker on an empty cell by `'w'`, and the worker on a destination cell by `'W'`,
- a box on an empty cell by `'b'`, and a box on a destination cell by `'B'`.

Part 1: data structures and output

Design data structures to represent a puzzle. Develop functions to show a configuration on the console using the above conventions. Use these data structures to represent the challenges that you find in the text files. Use the console output functions to verify that you correctly represented the challenges.

Part 2: sokuban, breadth first

Design and implement a *breadth-first search* algorithm to solve a given starting configuration. Test your implementation with the given challenges. Use the solutions to determine the least number of moves to solve these challenges.

Part 3: sokuban, depth first

Design and implement a *recursive*, *depth-first search* algorithm to solve a given starting configuration. Bound the depth of the search by an integer parameter. The algorithm must keep track of the *shortest-solution-so-far*, and abandon an attempt if its length exceeds the length of the *shortest-solution-so-far* or the given bound. Note that if the bound is too small, the algorithm will not find a solution.

4. Products

As product-to-deliver you only need to upload to Brightspace *"main.cpp"* that you have created with solutions for each part of the assignment.

⇒ **Deadline:** *Thursday, December 13, 2018, 13:30h.*

[1] https://en.wikipedia.org/wiki/Sokoban