Assignment 2: Charles the Robot reloaded

Course 'Imperative Programming' (IPC031)

Make sure to start working on the assignment **before the lab** session,
otherwise you will be too late for the deadline

1 Background

In this assignment you design and implement new algorithms for Charles in the same systematic manner that has been explained and demonstrated during the lecture.

2 Learning objectives

After doing this assignment you are able to:

- structure your algorithms by means of (parameterized) functions and control-structures.

3 Instruction

On Brightspace you find instructions to install the recommended programming environment VSCode and the Charles program that you need for this week.

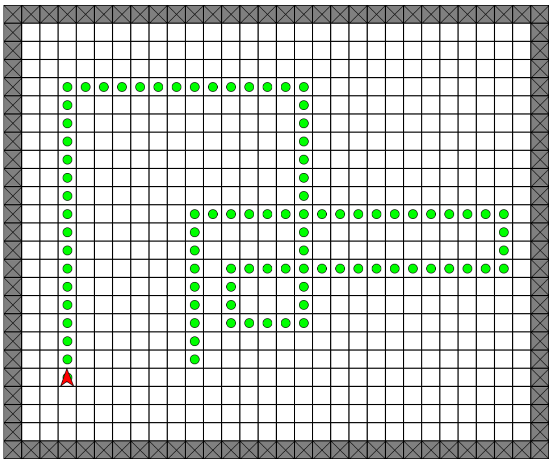
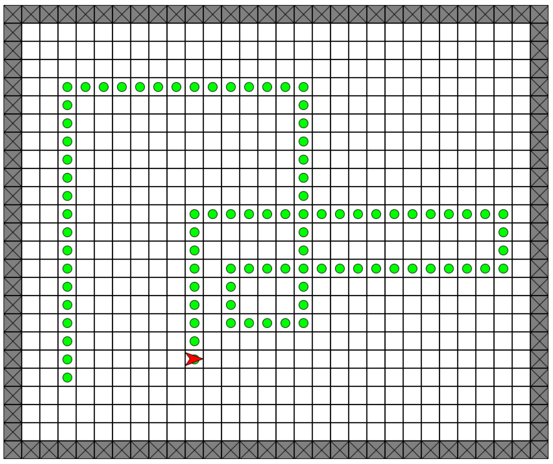
4 Assignment

In this assignment you use only the following C++ language constructs and Charles constructs that have been explained in the lecture:

- C++: sequence (`;`), choice (`if-else`), conditional repetition (`while`), logical operations; functions;
- Charles: `step`, `turn_left`, `turn_right`, `get_ball`, `put_ball`, `in_front_of_wall`, `north`, `on_ball`;
- functions to structure your code.

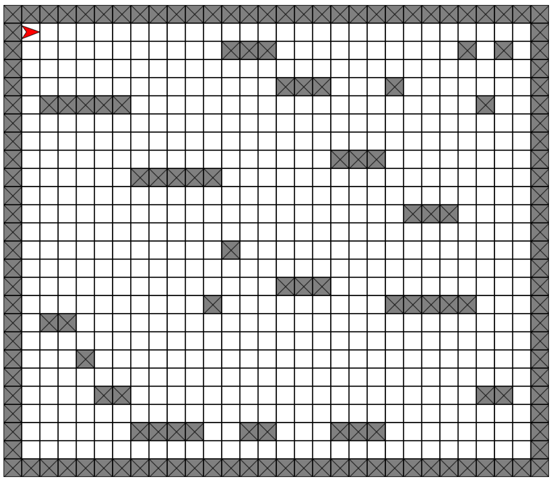
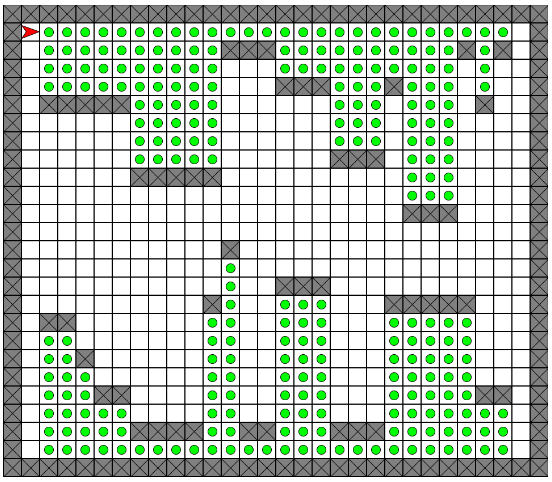
Below are the parts of the assignment that you are going to design and implement for Charles. You only need to modify "`agent.cpp`". The content of all other files does not concern this exercise. Be aware that function-definition must precede function-use in the source code.

Part 1: Hansl and Gretl

Adjust the function `path_agent ()` in "`agent.cpp`". This function can be invoked by the Charles command "Exercise 2.1: Hansl and Gretl". The effect of this function is that it makes Charles follow a contiguous path of balls such as the one shown in the above left figure until the last ball. Charles must end standing on top of the last ball and facing east, as shown in the above right figure. Your solution should work for any conceivable contiguous path of balls that do not contain *T-junctions*. You may assume that initially Charles is on the path somewhere and that its looking direction complies with the orientation of the path at that point.

Part 2: Stalactites and Stalagmites

Adjust the function `cave_agent ()` in "`agent.cpp`". This function can be invoked by the Charles command "Exercise 2.2: Cave". This command creates a world for Charles by adding a number of horizontally placed walls of random positions and widths. The figure gives an example. It is your task to make Charles create *stalactites* and *stalagmites*. A stalactite is a column of balls from the northern-most wall (ceiling) down to the first (horizontal) wall. A stalagmite is a column of balls from the southern-most wall (floor) up to the first (horizontal) wall. You can assume that except for the western-most wall and eastern-most wall and their immediate neighbor (vertical) streets, every other (vertical) street has 4 horizontal walls: the ceiling, the floor, and the two horizontal walls that delimit the stalactite and stalagmite.

Note: The Charles actions required for creating stalactites are identical to the Charles actions for creating stalagmites. In order to force you to write a symmetric algorithm, your `cave_agent ()` will automatically be executed twice in a row. Once for the stalactites, and once for the stalagmites. The second execution of your agent will continue where the first execution stopped. Make sure to design your algorithm for this agent keeping this in mind.

5 Products

As product-to-deliver you only need to upload to Brightspace the "`agent.cpp`" file that you have extended with solutions for each problem-solving-algorithm.

Deadline

Lab assignment: Friday, September 15, 2023, 23:59h