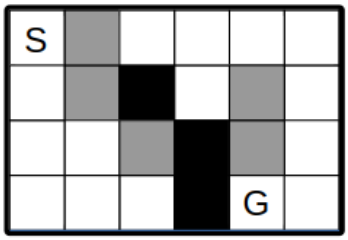
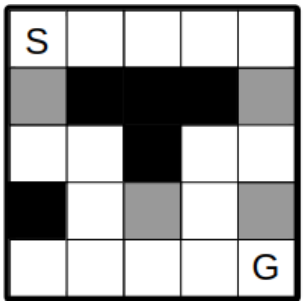
A* Search, Knowledge Representation, and Automated Planning

You should complete this coursework **individually**. Coursework 1 has **three parts** (A* search, knowledge representation in Prolog, and automated planning using PDDL) and is worth **17.5%** of your overall F29AI mark. Details of what you should do and hand in, and how you will be assessed, are described below and on Canvas.

Part 1: A* Search problem description





The above grids represent two problem environments where an agent is trying to find a path from the start location (S) to the goal location (G). The agent can move to an adjacent square provided the square is white or grey. The agent cannot move to a black square which represents a wall. No diagonal movement is allowed. Moving into a white square has a cost of 1. Moving into a grey square has a cost of 3.

What to do: Undergraduate students, 1-year Edinburgh postgraduate students, and all Dubai postgraduate students (full time and part time)

Implement a solution to the grid problem using A* search. You have two programming language options for this question: Java or Python. Starter code is provided for you in both languages.

Programming language option: Java

Starter code can be found in the package **uk.ac.hw.macs.search**, which is a set of classes that can be used to represent a search problem. To implement a specific search problem, you will need to do the following:

1. Define a class that implements the `State` interface. This should include the following:
    a. A method for determining whether a state is a goal state (`isGoal()`)
    b. A method for computing the heuristic value of a state (`getHeuristic()`)
2. Define a class that implements the `SearchOrder` interface. This interface has one public method, `addToFringe`, that is used to add a set of nodes to the frontier. You can use the costs and/or heuristic values to determine the order that nodes are added to the frontier

The classes in the **uk.ac.hw.macs.search.example** package show examples of these two interfaces being used to implement depth-first search and breadth-first search, as well as a simple integer-based state. The `Main` class in this package shows an example of how to use the classes.