Practical sheet: Pirate search

Part A : Pirate Grid World

1. Design (*not* implement!) a class for building a 2D, square grid world of different widths ranging from 3 to 8. For example, the *noughts and crosses* board is a 2D square grid of width 3, a *chess* board is a 2D square grid of width 8.

   The class should contain a constructor, destructor and an output() method. The output() method should draw the world by printing out a grid of characters or symbols.

   Each grid cell can be in one of 4 states: *unoccupied, occupied-by-treasure, occupied-by-obstacle, occupied-by-pirate*. You could represent this visually as follows:

```
-  -  -  -
-  O  -  O
T  O  -  -
-  O  P  -
```

2. Add to the constructor a means to:

   (a) Randomly position 4 Obstacles at *different* locations on the grid.
   (b) Randomly position 1 Treasure at an *unoccupied* position on the grid.
   (c) Randomly position 1 Pirate at an *unoccupied* position on the grid.

3. Devise algorithms (not software) for determining:

   (a) If there is an unobstructed path from the pirate to the treasure.
   (b) The length of the shortest path from the pirate to the treasure.

Part B : Implementation

4. Implement your design for questions 1 and 2.
5. Implement a random walk for the pirate that ensures it avoids obstacles and notifies the user when the treasure has been found.
6. Implement a simple algorithm that walks the pirate around performing a thorough and complete search, avoiding obstacles and notifying the user when the treasure has been found. **Hint**: keep track of which cells the pirate has visited during her search.

Part C : Advanced (optional)

7. Implement your algorithms for question 3.