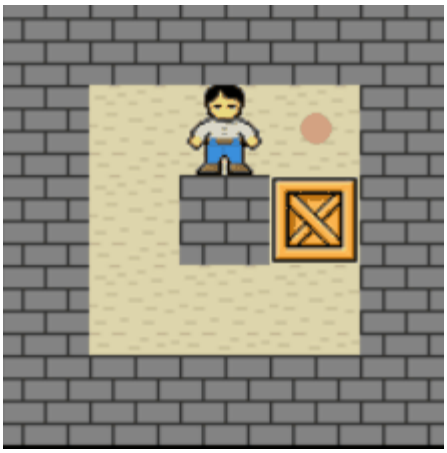
Project I: Push the Box

Push the box/Warehouse Keeper/Sokoban is a classic puzzle video game. The goal of the game is to push the box into the right position. In this project, we will program an agent to solve the puzzle.

Here is an example of the puzzle:



The agent controls the storage keeper (player), whose goal is to put the box(es) into the correct position (red dot). The map has walls, box(es), and space. The agent can move horizontally or vertically in four directions (up, down, left, right). While moving, the player can push a box into an empty space that is not a wall or another box. Notice that the agent can **only push the box, not pull the box.**

****

The input to the agent is the game map, which is described with several lines of characters, and stored in an input text file. Here is an example of the game map (the same in the previous picture):

```
#####
#.P*#
#.#B#
#...#
#####
```

Here, '#' stands for a wall, 'P' stands for the player, 'B' stands for the box, '.' stands for empty space, and '*' stands for the target place.

In this project, we will limit our scope to move **one box** to **one target position.** If you are interested, you can try to explore the scenario with more than one boxes.

****

What to submit (10pts):

- A report (4pts)
- Codes (3pts + 3pts + Extra) Q1.py and Q2.py.

- Follow the template.
- Do **NOT** change the program name and class/function name, or you may lose your grade.

Question 1: Basic search

In Question 1, we want to program an agent that returns the **minimum number of total moves that the player** needs. Repeat that for a single move, the player can go up, down, left or right.

Input format: A text file that includes the game map. In Q1, the size of the map is limited to at most 20x20.

Output format: Your program should return a single integer that indicates the total number of moves needed for winning the game. **If the player cannot win, you should return -1.**

Question 1.1 (1pts): Shall we choose DFS or BFS for the task? Why?

Question 1.2 (1pts for description + 3pts for code): Following code template Q1.py, finish the search of your answer in Q1.1. Briefly describe how you design your programs.

- **After submission, your code will be tested against 10 test inputs (0.3pts each). If your code gives the correct answer within 5 seconds, you will get the points.**
- **After finishing your code, please first test your code with the test inputs in Q1 – Follow the guidelines in Q1.py. Beware the corner cases!**

Question 2: Heuristic search

In Question 2, we want to program an agent that returns the **minimum number of total pushes that the player** needs. One move of the box is considered a push. **(Notice the difference to Q1!)**

Input format: A text file that includes the game map. In Q2, the size of the map is extended to 50x50, and the size of extra credit map is 100x100.

Output format: Your program should return a single integer that indicates the total number of pushes needed for winning the game. If the player cannot win, you should return -1.

Question 2.1 (1pts): Design an admissible heuristic function for the task.

Question 2.2 (1pts for description + 3pts for code): Following code template Q2.py, complete an A* algorithm with the heuristic of your choice in Q2.1. Briefly describe how you design your programs.

- **After submission, your code will be tested with 10 test inputs (0.3pts each). If your code gives the correct answer within 5 seconds, you will get the points.**

- **Extra credit (1-3pts): Your code will be tested against 5 extra large** (100x100) **maps. All the submitted codes in the class will be tested together and rank by the number of map solved, and then the total time spent as tie breaker.**
    - **If your program gets top 50% in the class, you will get 1pts extra credit.**
    - **If your program gets top 20% in the class, you will get 2pts extra credit.**
    - **If your program gets top 10% in the class, you will get 3pts extra credit.**
- **After you finished your code, please first test your code with the test inputs in Q2 - Follow the guidelines in Q2.py. Similar to Q1, beware the corner cases!**