Programming Assignment 1 — Bomberman
Algoritms, Spring 2020

In the first assignment, we will work with a simplified version of a game, Bomberman. In particular, we want to know, given a world consisting of walls (that are indestructible) and crates (that are destructible), how many bombs we would need to destroy all crates.

State-spaces

The Bomberman world is a 2D world, that can be represented by a Numpy ND-array. Figure 1 shows an example board.

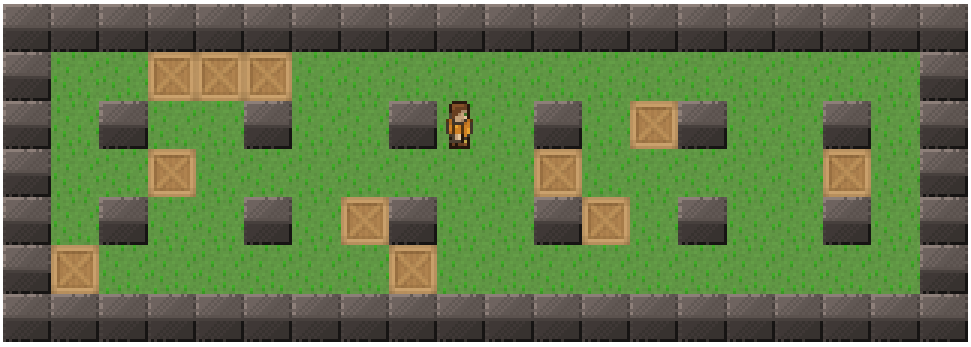

Figure 1: Example board.

The gray squares represent walls, the brown squares represent crates, and the green squares represent empty space. The player can move over the empty spaces, from each position, he/she can reach the ortogonal adjacent squares (left, right, above, below, but not diagonal). Note that in the board in Figure 1, not all empty squares are reachable: the green squares at the top left are blocked by walls and crates. Hence, first some crates need to destroyed before these can be reached.

By placing bombs, the player can destroy crates. The player can place a bomb on his/her position, and the explosion will traverse in all ortogonal directions (left, right, top, bottom) until it hits a crate or a wall. Due to this, a single bomb can destroy at most four crates. If it hits a crate, the crate will be destroyed. If it hits a wall, the wall will remain. Figure 2 shows an example of an explosion.

The above examples show worlds that have a border of walls on all sides. This is not neccessarily the case. The border can also contain crates or empty spaces. Explosions can not traverse beyond the border.

Problem Statement

The goal is to write two algorithms:

- An optimal algorithm (brute-force), which determines exactly the minimal number of bombs required to destroy all crates. Use recursion to write this algorithm. As shown in the lectures, there are two ways of achieving this, by copying the game state into a new instance of the data structure, and by implementing the do move / undo move paradigm. Select which one you prefer.

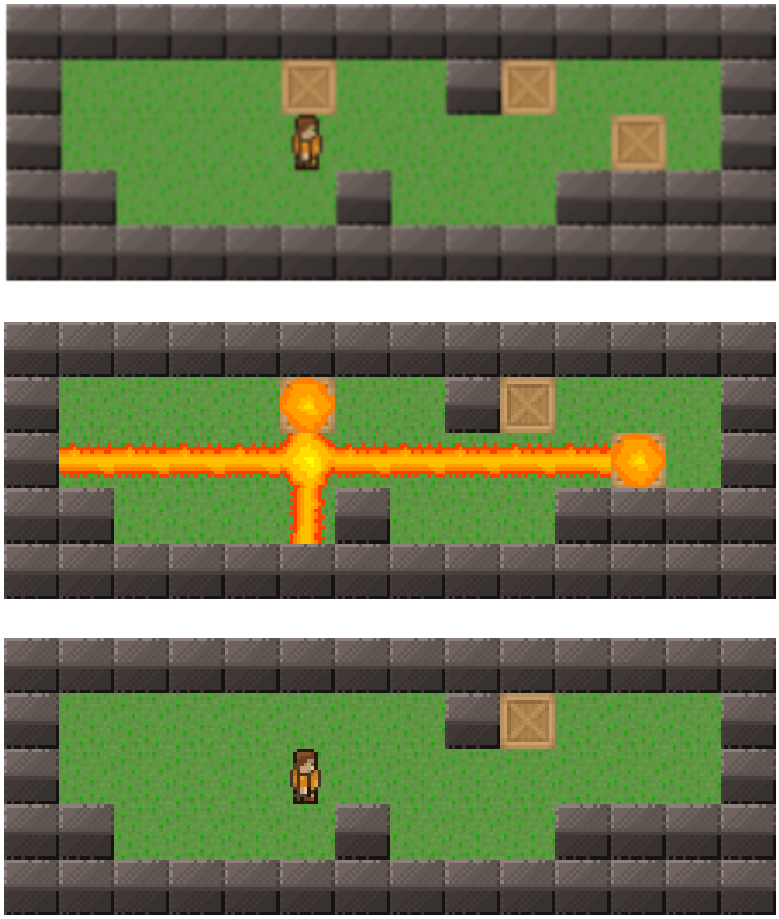

Figure 2: An example of a bomb, destroying two crates.

- A greedy algorithm, that at any position determines at which place a bomb could destroy the maximum number of crates. You do not need recusion for this function. It is probably easier to implement, so maybe start with this function.

There are two components that make this assignment complicated:

- Specific parts of the board may not be reachable initially (see Figure 1). By making use of the floodfill algorithm, the positions that are reachable from a given position can be determined.
- Several moves result in the same state, as illustrated by Figure 3. It is important to detect which of these moves are equivalent, and only try a single one of these.

Templates

There is a template program on the Blackboard page. It consists of two files:

- `assignment1.py`: The main assignment file.
- `test_public.py`: A Unit Test file.

Figure 3: Example of equivalent moves. Placing a bomb on position 1 t/m 7 is equivalent. Also placing a bomb on position 8 and 9 is equivalent.

Additionally, we held back a file called `test_private.py`, which will be used to grade the assignment. This one will be published after the deadline has passed.

The file `assignment1.py` contains various functions, that are not implemented yet (in total: 8). Read the written documentation about these functions, and implement these functions. Most of these functions require less than 15 lines of code. None of the functions require more than 30 lines of code. Do not change the headers of the functions provided. You are allowed to add functions yourself, if you feel that that makes it easier.

The file `test_public.py` consists of several test functions. We provided these functions to help programming and testing the code. By running the following command, the unit tests can be executed:

```
python -m unittest test_public.py
```

Make sure to have the right packages installed (numpy). Feel free to add more unit tests. Do not alter the unit tests that are provided. If your program does not succeed on all unit tests that are provided, it is likely that there is still a problem in your code. If you are not able to detect the equivalent moves, two unit tests will fail, but you can still obtain a passing grade. (Make sure that all other unit tests succeed, before submitting the code).

Report

Write a report in LaTeX(at most 3 pages), addessing the following points / research questions:

- Introduction: describe the game and problem. Describe a state and action.
- Draw the state-space of a small example, e.g., on a 5×5 board. You are allowed to draw this on paper and scan the result.
- Analysis of optimal method: How much time does the optimal method take on boards of varying size? How fast does it run on a board of 7×7, and how fast does it run on a board of 10×10? What are the limits that can still terminate within 10 minutes?
- How well does the greedy approach work? How often and how much does the answer differ from the result of the optimal algorithm? Include a test-case in your report where the difference between the greedy algorithm and the optimal algorithm is as high as possible on a board smaller than 10×10.
- Summary and Discussion. Always end a report with some summarizing remarks.

Submission

Work in pairs of 2, and upload your submission to Blackboard before the deadline. Use the team submission for this, so it is clear which students have worked on it. Upload a zip file, with **only** the following three files:

- `assignment1.py`, having filled in all functions
- `report.pdf`, report written in LaTeX
- `test_public.py`, potentially adding the unit tests you designed yourself

In particular, Apple users, make sure that there are no additional 'rubbish' files in the zip. Make sure that the code runs on any LIACS computer with Python3 (test in room 304/306).

Deadline: March 23rd, 9:00. There is an option to hand this in after the deadline (up to at most 2 weeks), however a point will be substracted from the grade for every (part of a) week that it is handed in late.