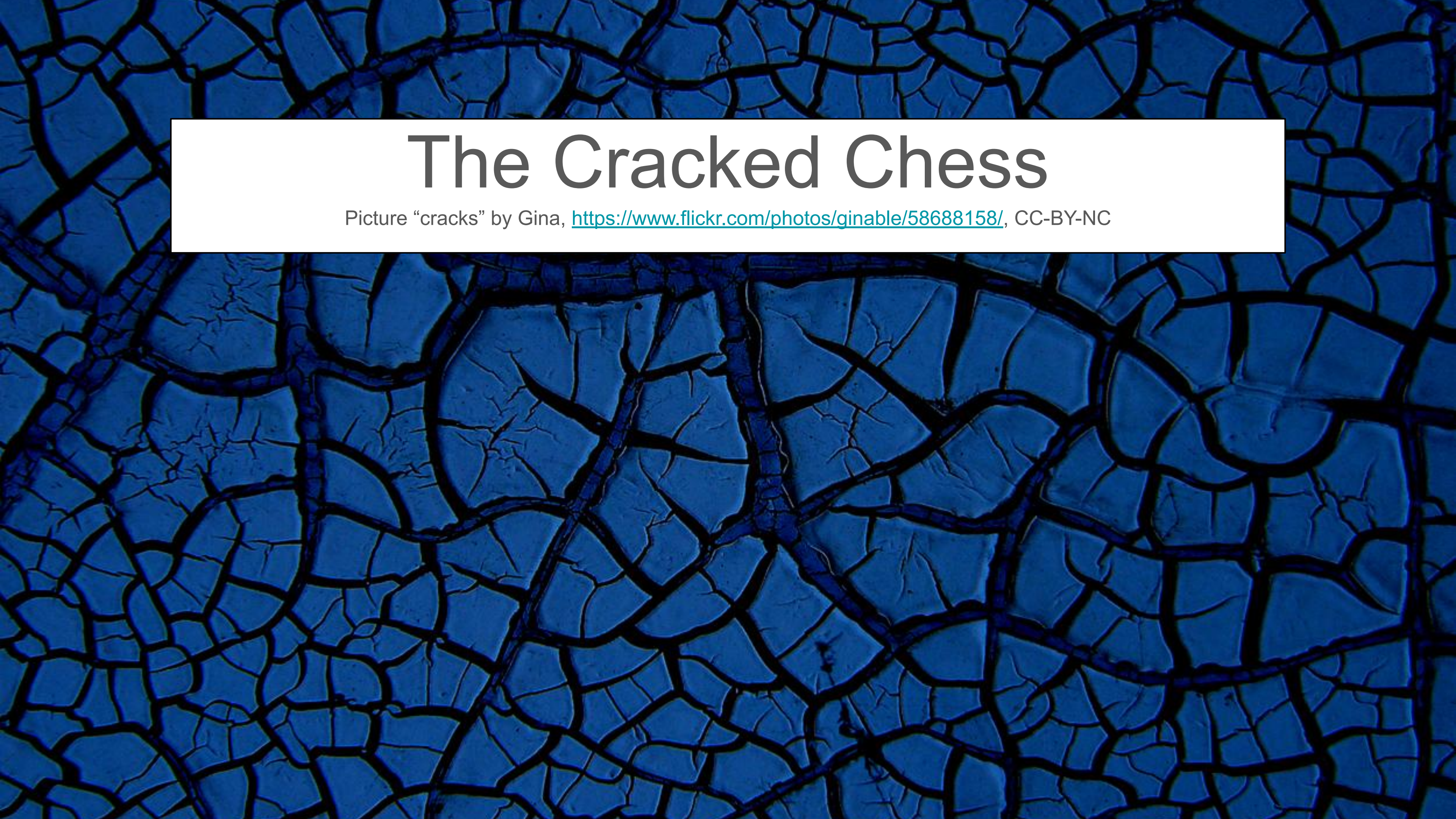
Workshop

Computational Thinking and Programming 22/23

Silvio Peroni

Digital Humanities Advanced Research Centre (/DH.arc)
Department of Classical Philology and Italian Studies, University of Bologna, Bologna, Italy
silvio.peroni@unibo.it – @essepuntato – https://www.unibo.it/sitoweb/silvio.peroni/en

ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA
DEPARTMENT OF CLASSICAL PHILOLOGY
AND ITALIAN STUDIES



The Cracked Chess

Picture “cracks” by Gina, https://www.flickr.com/photos/ginable/58688158/, CC-BY-NC

Plot

In the past, great explorers discovered the path to reach one of the most mysterious place in the world and beyond, which contained some ancient collections of the Library of Babel. Among these, a book was actually catching all the thoughts of such explorers, i.e. the *Book of Indefinite Pages*, containing all the possible books ever written in just one portable volume.

However, to bring back the Book, the explorers had to pass through one of the most perfidious artifacts created by Myntrakor, also known as Who Must Not Be Thought and ruler of that mysterious place. That artefact was the Tower Labyrinth, the darkest meta-dimension known to human beings, made of 100 squared mazes of different dimensions placed one upon the other.

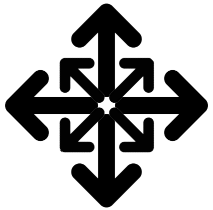
Only a few of the explorers were able to go out from that system of labyrinths, and we are so lucky to have had the chance to read the marvellous content described in the Book.

Thus we are all here, drinking our beverages at the World End’s Inn, a flourishing tavern across worlds, to pass a bit of time playing one of the games introduced in the Book: The Cracked Chess – a.k.a. Our Beloved Queen.

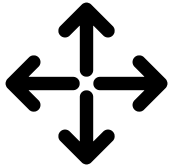
Pieces of the game

Three kinds of pieces:

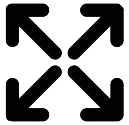
Queen moves:

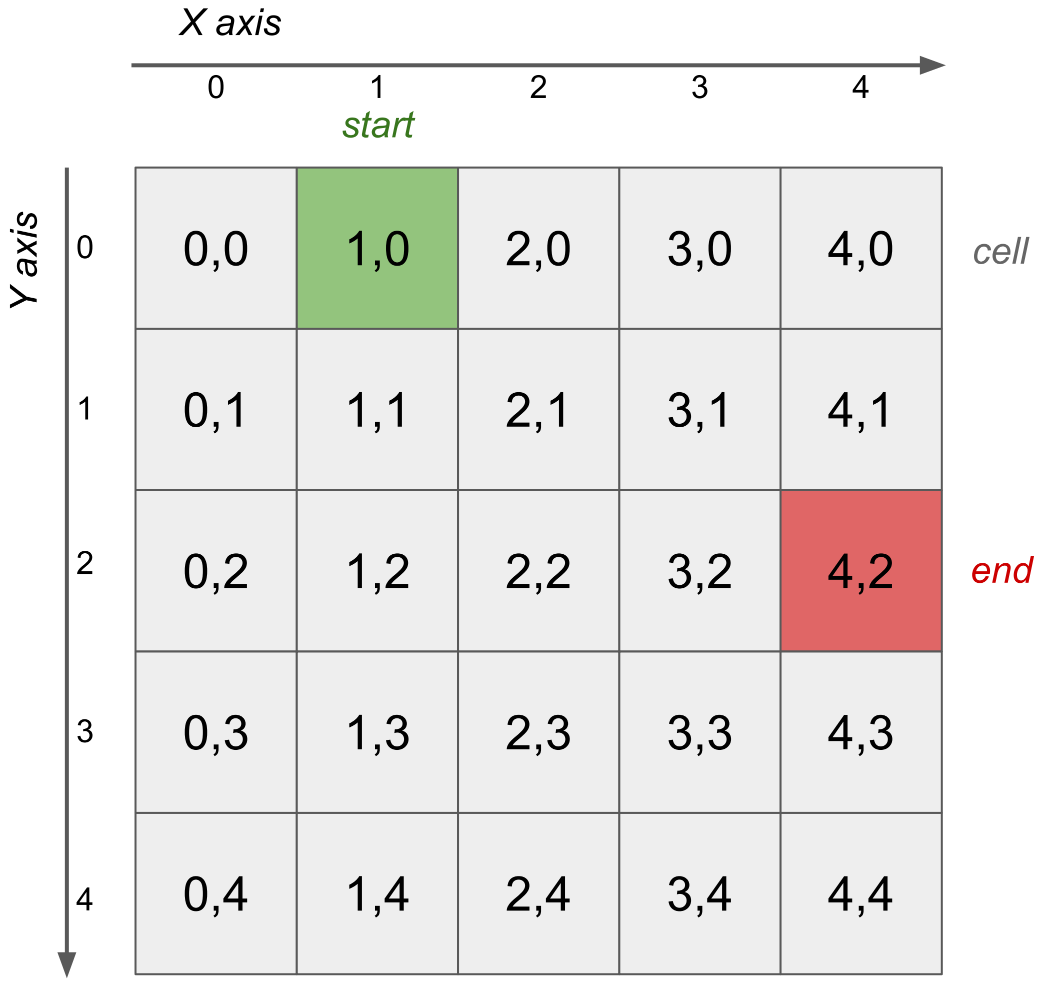


Rock moves:



Bishop moves:





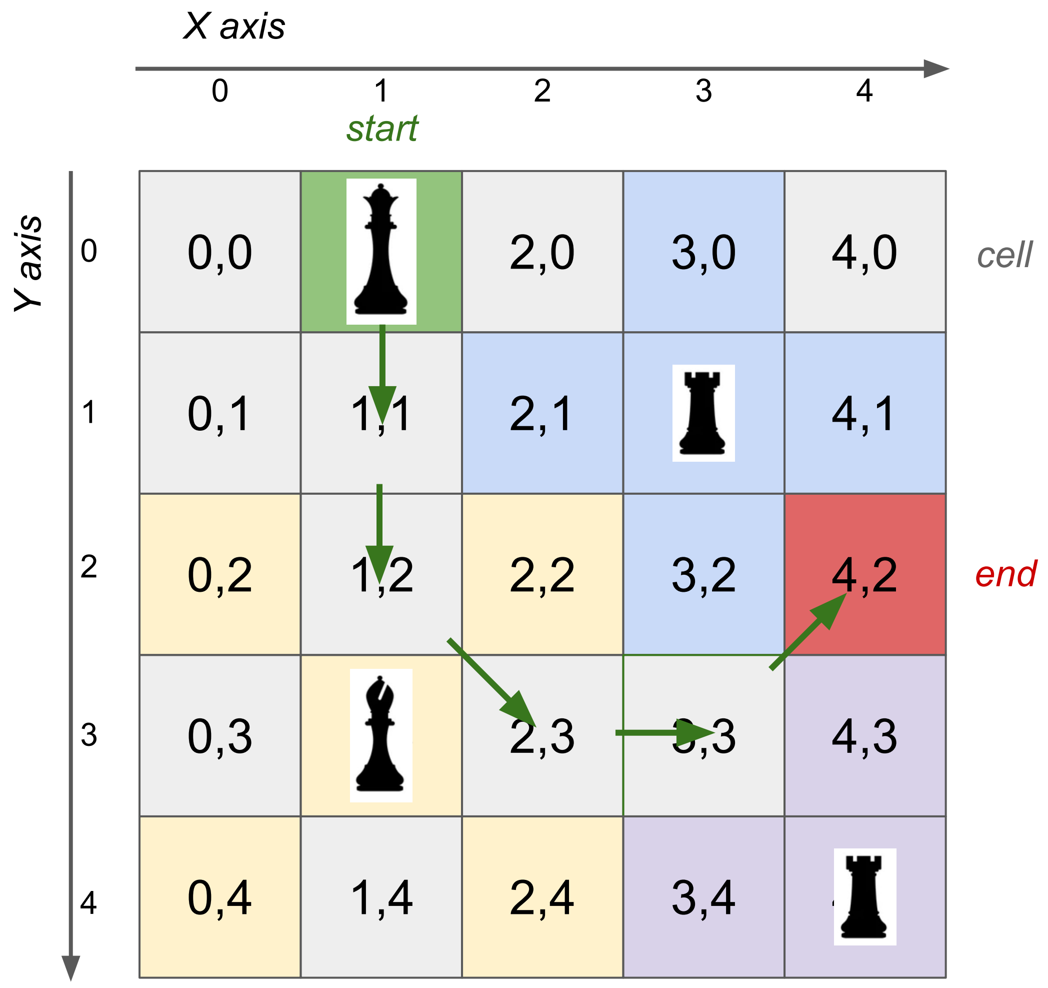
Rules

1. The board is a square which is initially **filled with an arbitrary number** of rocks / bishops
2. The player **controls the queen**, initially positioned in the start cell
3. The board is cracked and, thus, all pieces in the board can move **only of one cell per turn**, according to their possible movements (see previous slide)
4. The rocks and bishops **do not move unless** they can capture the queen
5. The queen **must always move** to a close cell in each turn
6. The queen **cannot capture** any piece and cannot move on a cell already occupied
7. The **goal of the player** is to move the queen, turn after turn, to **reach the exit cell**
8. The queen is captured (i.e. **the player lose the game**) if
   a. the queen do not move
   b. the queen moves to a place which is not an adjacent cell, a cell that is controlled by a bishop/rock, and a cell which is not contained in the board
   c. the queen does not reach the exit within a maximum number of turns that depends on the dimension of the board (4 turns in a 2x2 square, 9 in a 3x3, 14 in a 4x4, etc.)

Example

If the queen moves in one of the cells controlled by the other pieces (blue, yellow, purple cells), it will be captured in the following turn by them

The queen can move of one cell per turn, in any direction



Function to implement

```
def do_move(board, current, exit, notebook)
```

It takes in input:

- `board`, a list of lists of dictionaries representing the rows of the board and the cells in each row
- `current`, a tuple of X/Y coordinates identifying the current position of the queen in the board
- `exit`, a tuple of X/Y coordinates identifying the exit cell
- `notebook`, a dictionary (empty in the first turn) available for note taking about a particular game

It returns a tuple of two items:

- the first item contains a tuple defining the X/Y coordinates of the next adjacent cell to move the queen to – e.g. `(1,1)`
- the second item contains the notebook that can be modified with additional information as a consequence of the choice of the move in the first item

Example of a board

```
[               The main list defining the board
    [               The first row (i.e. a list) of the board
        {   'x': 0,                 X coordinate of the first cell of the first row
            'y': 0,                 Y coordinate of the first cell of the first row
            'type': 'free' },       type of cell (‘empty’, ‘rock’ or ‘bishop’)
        {   'x': 1,                 X coordinate of the second cell of the first row
            'y': 0,                 Y coordinate of the second cell of the first row
            'type': 'rock' }, ...   type of cell (‘empty’, ‘rock’ or ‘bishop’)
    ],
    [ … ], …    The second row (i.e. a list) of the board
]
```

How it works

The file responsible for showing how to run the game is **`run.py`**

The initial part of the file **`run.py`** contains an exemplar 4x4 board with one rock and one bishop in two distinct cells, followed by the starting cell and the exit cell

The function **`do_move`** is included in the file **`group.py`** and will be called in the body of the while loop in **`run.py`** – please note that the code in **`run.py`** does not check if the move is valid or not

To run your implementation of **`do_move`** you can run

```
python run.py
```

(The initial implementation of **`do_move`** will result in running **`run.py`** indefinitely)

Points to the groups

1 point will be assigned to all groups that developed a function that allowed their queen **to avoid wrong moves**, i.e. not moving or moving in a cell that is not adjacent of queen’s current position, a cell that is controlled by a bishop/rock, or a cell which is not contained in the board

1 point will be assigned to the group that **won the greatest number of games** (i.e. its queen reached an exit before the others overall) out of 100 games

1 point will be assigned to all groups that developed a function that allowed their queen to find the exit of **at least 30** different boards

1 point will be assigned to all groups that developed a function that allowed their queen to find the exit of **at least 90** different boards

Material and submission

All the files are available at
https://github.com/comp-think/2022-2023/tree/main/docs/workshop

At the end, each group must send an email to silvio.peroni@unibo.it where are specified:

- The name of the group
- The members of the group
- The file containing the implementation of `do_move` (rename it from `.py` to `.txt` before attaching it to the email)

End

Workshop

Computational Thinking and Programming 22/23

Silvio Peroni

Digital Humanities Advanced Research Centre (/DH.arc)
Department of Classical Philology and Italian Studies, University of Bologna, Bologna, Italy
silvio.peroni@unibo.it – @essepuntato – https://www.unibo.it/sitoweb/silvio.peroni/en

ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA
DEPARTMENT OF CLASSICAL PHILOLOGY
AND ITALIAN STUDIES