Workshop

Computational Thinking and Programming 23/24

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
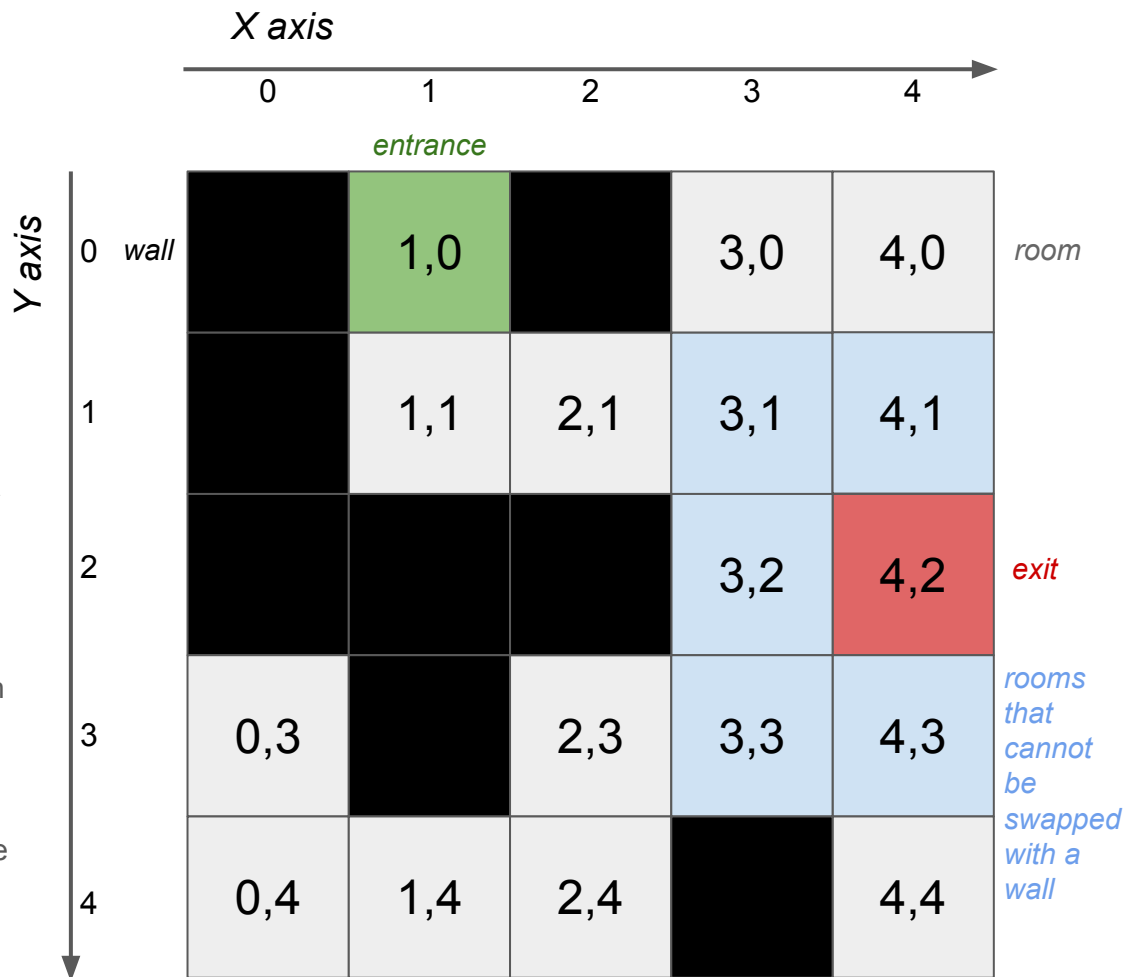AND ITALIAN STUDIES

Moving walls

Plot

Myntrakor, also known as Who Must Not Be Thought, was cheated! A mere human explorer was able to survive to the Tower Labyrinth, a complex system made of 100 squared mazes of different dimensions placed one upon the other created to custody one of the most marvellous gems of the (now destroyed) Library of Babel: the Book of Indefinite Pages, containing all the possible books ever written in just one portable volume.

The content of the Book was unbelievable. However, mere humans used it just to take inspiration for the creation of new abstract strategy games – fools who do not recognise the immense power that the Book encloses!

Myntrakor, dressed up as a player of the game Cracked Chess (very popular with the customers of the World End's Inn), accessed the Inn, found the Book and took it to bring it back within the Tower Labyrinth. However, to avoid others having the chance to steal the Book again, he hired Urg, the best human player of Cracked Chess, a despicable man, after all, as a guarantor of the Tower Labyrinth. Myntrakor asked him to implement a new innovative idea for each maze of the Tower, i.e. a mechanism that allows the walls to move while someone is trying to reach the exit…
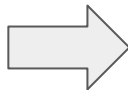
Structure of a maze

1. Each maze is organised within a square of rooms with no roof, with an entrance room and an exit room
2. An explorer can move horizontally or vertically through the rooms
3. All moves are organised in turns; in every turn:
   (a) **The explorer,** which has a system to take a bird's-eye view screenshot of the labyrinth, moves to an adjacent room, following the shortest path to reach the exit
   (b) **Urg** swaps a wall of the labyrinth with an available room, to prevent the explorer to reach the exit
4. Urg wins if he prevents the explorer to reach the exit in a limited amount of turn, i.e. the double of the length (in rooms/walls) of an edge of the labyrinth, otherwise he will be punished by Myntrakor

X axis

	0	1	2	3	4	
		entrance				
0 *wall*		1,0		3,0	4,0	*room*
1		1,1	2,1	3,1	4,1	
2				3,2	4,2	*exit*
3	0,3		2,3	3,3	4,3	*rooms that cannot be swapped with a wall*
4	0,4	1,4	2,4		4,4	

Y axis

Urg moves per turn

Rules

1. Each labyrinth is a square (of length n per edge) which is initially **filled with an arbitrary number** of rooms and walls
2. The explorer, in every turn, try to **move in a room** (vertically or horizontally) to get closer to the exit, initially positioned in the start cell
3. In case, in a turn, the explorer see that there is no path available to reach the exit, he **uses the teleport** that will bring him in a **random room** (the exit room could be accidentally selected!)
4. The player (**you!**) control Urg and, thus, decides, in every turn, **which wall should be swapped with which room** (it is a mandatory move)
5. The player **cannot put a wall** in the exit room, in any (vertical, horizontal, diagonal) adjacent room of the exit room and in the room currently occupied by the explorer
6. The **goal of the player** is to avoid that the explorer reaches the exit within $n * 2$ moves

Function to implement

```
def do_move_wall(labyrinth, length, explorer_position, exit, notebook)
```

It takes in input:

- **labyrinth**, a list of tuples of X/Y coordinates representing the rooms of the labyrinth
- **length**, a number identifying the length of the edge of the labyrinth
- **explorer_position**, a tuple of X/Y coordinates identifying the current position of the explorer
- **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 the new configuration of the labyrinth (i.e. the list of cells defining it) after having moved the selected wall
- the second item contains the notebook that can be modified (if needed, it is not mandatory to modify it) with additional information as a consequence of the choice of the move in the first item

Example of a board

```
[

(0,0),(3,0),(4,0),(7,0),(
1,1),(2,1),(4,1),(6,1),(7
,1),(1,2),(5,2),(6,2),(7,
2),(2,3),(3,3),(4,3),(5,3
),(6,3),(7,3),(0,4),(1,4)
,(2,4),(3,4),(6,4),(7,4),
(0,5),(2,5),(0,6),(1,6),(
2,6),(4,6),(5,6),(6,6),(0
,7),(1,7),(4,7)

]
```
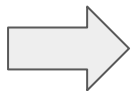
```
[

(0,0),             (3,0),(4,0),             (7,0),
     (1,1),(2,1),      (4,1),      (6,1),(7,1),
     (1,2),                  (5,2),(6,2),(7,2),
          (2,3),(3,3),(4,3),(5,3),(6,3),(7,3),
(0,4),(1,4),(2,4),(3,4),             (6,4),(7,4),
(0,5),      (2,5),
(0,6),(1,6),(2,6),      (4,6),(5,6),(6,6),
(0,7),(1,7),             (4,7)

]
```

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 8x8 labyrinth with a starting cell and the exit cell, and the algorithm implementing the moves of the explorer has been already implemented and provided (file `explorer.py`)

The function `do_move_wall` (you have to implement!) 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 the wall swap is valid or not

To run your implementation of `do_move_wall` you can run
`python run.py`

(The initial implementation of `do_move_wall` will result in a mistake)

Points to the groups

1 point will be assigned to all groups that developed a function that **avoids wrong wall-room swaps**, i.e. putting either a wall or a room outside the labyrinth, putting a wall in the current position occupied by the explorer, and putting a wall in any adjacent (vertical, horizontal, diagonal) rooms of the exit room

1 point will be assigned to the group that **won the greatest number of games** (i.e. killing the explorer) out of 100 distinct labyrinth

1 point will be assigned to all groups that developed a function that allowed Urg to kill the explorer in **at least 30** different labyrinth

1 point will be assigned to all groups that developed a function that allowed Urg to kill the explorer in **at least 70** different labyrinth

Material and submission

All the files are available at
https://github.com/comp-think/2023-2024/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_wall` (rename it from `.py` to `.txt` before attaching it to the email)

End

Workshop

Computational Thinking and Programming 23/24

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